Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

InDrone: Progettazione e Sviluppo di un Frontend Web per la Gestione delle Missioni di UAV in Ambienti Indoor

Relatore:
Chiar.mo Prof.
DI IORIO ANGELO

Correlatore:
Dott.
MONTECCHIARI
LEONARDO

Presentata da:
IAMONACO MICHELA

I Sessione
Anno Accademico 2023/2024

Sommario

In questa tesi viene trattato il processo di progettazione e implementazione di un'applicazione web per la creazione e il monitoraggio di missioni effettuate con droni in un ambiente indoor. Dopo uno studio dello stato dell'arte, sono stati analizzati i software esistesti per identificare i punti di forza e debolezza, così riuscendo a definire i requisiti per l'applicazione.

L'utilizzo di un metodo *agile* per la progettazione e sviluppo dell'applicazione ha permesso un collaborazione e interazione ottimale con il Team di Robotica del Technology Innovation Institute (TII). Attraverso l'uso di mockup abbiamo delle interfacce funzionali e user-friendly, che rispondessero ai requisiti. Anche la fase di implementazione è stata caratterizzata da un progresso iterativo e collaborativo, garantendo adattabilità e miglioramento delle funzionalità.

Il risultato è ***InDrone***, un'applicazione web validata attraverso dei test condotti dal Team di Robotica del TII, che soddisfa i requisiti inizialmente definiti.

Introduzione

La crescente diffusione dell'uso dei droni ha suscitato un significativo interesse nello sviluppo di soluzioni avanzate che possano sfruttare queste tecnologie in contesti indoor per gestire e monitorare missioni, nonchè per eseguire compiti che risulterebbero difficili o pericolosi per l'uomo. I droni, infatti, sono in grado di svolgere tali operazioni in modo efficiente e sicuro, aprendo nuove possibilità nel campo della robotica e dell'automazione.

Questa tesi ha come obiettivo principale la progettazione e l'implementazione di un'applicazione web, che consenta la creazione e il monitoraggio di missioni effettuate da droni in un ambiente indoor.

Inizialmente, è stato condotto uno studio dettagliato dello stato dell'arte, partendo dai fondamenti della robotica e focalizzandosi specificamente sui droni per comprenderne la struttura e il funzionamento. Successivamente, sono stati analizzati software esistenti che potevano avvicinarsi alla soluzione desiderata, permettendoci di identificare punti di forza e di debolezza di queste tecnologie e di definire chiaramente i nostri requisiti.

Il processo di progettazione dell'applicazione ha visto una definizione accurata dei requisiti funzionali e non funzionali in collaborazione con il Team di Robotica del TII. Attraverso l'elaborazione iterativa di mockup, si è mirato a creare un'interfaccia intuitiva e funzionale, che rispondesse alle esigenze operative.

Durante la fase di implementazione dell'applicazione, l'adozione di metodologie di sviluppo *agile* ha consentito un progresso iterativo e collaborativo, facilitato da incontri regolari con il Team di Robotica, nei quali venivano va-

lutati i risultati e aggiornati gli obiettivi di sviluppo. Questo approccio ha garantito una rapida adattabilità ai cambiamenti e un costante miglioramento delle funzionalità dell'applicazione.

Ed è così che infine nasce **InDrone**, un'applicazione web per la creazione e monitoraggio di missioni di droni indoor.

Una volta completata, **InDrone** è stata validata attraverso dei test effettuati dal Team di Robotica del TII, evidenziando che l'applicazione sviluppata soddisfa i requisiti inizialmente identificati. Nonostante i successi raggiunti, sono emerse alcune limitazioni, come la mancanza di integrazione server-side e la possibilità di implementare funzionalità avanzate come lo streaming video in tempo reale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Analisi Stato dell'Arte

1.1 Cenni Storici della Robotica

La robotica è lo studio e lo sviluppo di metodi che consentono ai robot di svolgere compiti specifici, replicando automaticamente il lavoro umano. Sebbene la robotica sia una branca dell'ingegneria, più precisamente della meccatronica, riunisce approcci di molte discipline, comprese quelle umanistiche come la linguistica e le discipline scientifiche come la biologia, la fisiologia, la psicologia, l'elettronica, la fisica, l'informatica, la matematica e la meccanica.

La definizione di robot è stata affinata nel tempo. Nel 1929 la *Treccani* definiva un robot come una creatura meccanica in grado di eseguire, in forma automatica, azioni e operazioni che riproducono quelle umane, mentre oggi viene definito come un apparato meccanico ed elettronico programmabile, impiegato nell'industria, in sostituzione dell'uomo, per eseguire automaticamente e autonomamente lavorazioni e operazioni ripetitive, o complesse, pesanti e pericolose.

1.1.1 Origini e prime applicazioni

Da secoli l'umanità è affascinata dal concetto di robot, immaginando robot umanoidi che possano aiutarci e svolgere compiti al posto nostro. Già

alla fine del XV secolo Leonardo Da Vinci, guidato dai suoi studi anatomici effettuati per l'Uomo Vitruviano, progetta l'*Automa Cavaliere*, un robot umanoide in grado di alzarsi in piedi, muovere la testa, la mascella e le braccia. Solo nel 1950 Carlo Pedretti, con l'aiuto di altri studiosi, riuscì a realizzare una copia fedele dell'Automa Cavaliere studiando i disegni presenti nel Codice Atlantico e in altri taccuini. Questo automa si presenta come un soldato rivestito da un'armatura medievale, capace di compiere alcuni movimenti fisici di base, come alzare il capo oppure un arto. I movimenti sono resi possibili grazie a un rudimentale sistema meccanico posizionato nelle parti superiori, con un sistema di cavi, e inferiori, con rotelle. All'interno è costruito in legno, con elementi in pelle e metallo.

Figura 1.1: Modello dell'automa cavaliere di Leonardo e (a fianco) i suoi meccanismi interni (esposizione Leonardo da Vinci. Mensch - Erfinder - Genie, Berlino 2005)

L'Automa Cavaliere può essere considerato il primo robot umanoide della storia. Però non sappiamo se Da Vinci abbia provato a costruirlo; perciò, per avere il primo robot concreto della storia dobbiamo aspettare il 1738 quando Jacques de Vauncason fabbrica un androide capace di suonare un flauto, mentre nel 1774 l'inventore dell'orologio da polso, Pierre Jacquet-Droz, fabbricò The Scribe, un robot antropomorfo ispirato ad un bambino

di tre anni che seppe sorprendere per il fatto di saper seguire con lo sguardo un testo durante la sua scrittura.

Nonostante questi sviluppi, i termini "robotica"e "robot"non erano ancora stati inventati. Come raccontato Rick Davenport nel suo capitolo Robotics[2], solo nel 1920, Karel Capek, uno scrittore ceco, coniò la parola "robot"nella sua opera teatrale intitolata "Rossum's Universal Robots", che rappresentava robot avanzati con un aspetto umanoide che si ribellavano. Invece nel 1942 vediamo Isaac Asimov usare il termine "robotica"nel suo racconto breve intitolato "Runaround", in cui introduce anche le sue tre leggi della robotica, che miravano a garantire la sicurezza umana. Asimov vedeva la tecnologia in modo ottimistico, a differenza di Capek. Oggi, sebbene non abbiamo ancora raggiunto il livello di intelligenza artificiale immaginato da Asimov, i ricercatori stanno esplorando le implicazioni sociali della progettazione dei robot.

La robotica si è evoluta da dispositivi con reazioni limitate a robot con capacità decisionali indipendenti. Questa evoluzione ha permesso alla robotica di espandersi in numerosi settori. Nell'industria, i robot sono utilizzati per la costruzione di automobili e altre applicazioni manifatturiere. In medicina, robot avanzati assistono i chirurghi durante interventi complessi. Nel settore militare, i robot vengono impiegati per la sorveglianza e la difesa. I robot esplorativi vengono utilizzati per filmare vulcani attivi o esplorare pianeti lontani. Inoltre, i robot vengono utilizzati per compiti pericolosi, come testare ambienti ostili o visitare siti nucleari, e nella cura delle persone, assistendo nella terapia e riabilitazione.

1.2 Evoluzione della Robotica

A metà degli anni '50 viene introdotto il primo robot sviluppato per automatizzare alcune attività industriali, "Unimate", un braccio robotico programmabile realizzato da Goerge C. Devol e Joseph Engleberger. So-

lo nel 1962 il primo modello di Unimate fu venduto alla Generale Motors[1] per lavorare nei compiti di carico e scarico con macchine a pressofusione riscaldate, operazioni molto pericolose invece per l'essere umano a causa dei potenziali danni fisici e fumi tossici rilasciati. Successivamente Engleberger fondò la prima azienda commerciale per la produzione di robot industriali, la Unimation, guadagnandoli il titolo di "Padre della robotica".

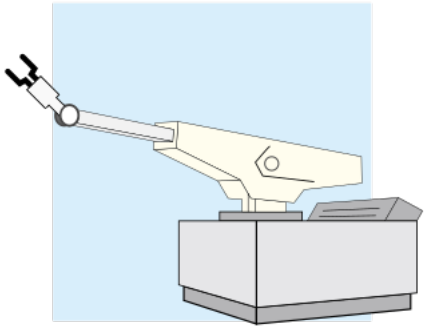


Figura 1.2: Schematizzazione del modello Unimate robot

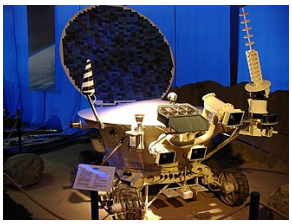
In seguito, la robotica si sviluppa ulteriormente diffondendosi al di fuori del settore industriale.

Negli anni '70 durante la missione spaziale Luna17 fu utilizzato per la prima volta un rover lunare robotico controllato a distanza da operatori sulla Terra, il Lunokhod-1.

Questo traguardo ha inaugurato nuove prospettive nella robotica spaziale, dimostrando che i robot potevano essere utilizzati con successo per l'esplorazione dello spazio. Questo rappresenta un avanzamento significativo che ha ispirato e orientato lo sviluppo di tutti i successivi rover spaziali, come Spirit, Opportunity e Curiosity della NASA.

Nel 2000, Honda introdusse ASIMO, uno dei primi robot androidi, considerato il primo esempio di robot con sembianze umane. ASIMO poteva camminare, correre e interagire con l'ambiente circostante. Inoltre, fu il primo modello a integrare il controllo predittivo dei movimenti, un sistema

[1] https://www.gm.com/



(a)

(b)

Figura 1.3: Le immagini sopra raffigurano: (a) Lunokhod-1, rover lunare robotico e (b) Robot androide ASIMO

che consente al robot di anticipare i movimenti successivi e di modificare il proprio centro di gravità, migliorandone così l'equilibrio e la flessibilità.

Qualche anno dopo, i robot diventano accessibili al grande pubblico grazie all'azienda americana iRobot, che introduce Roomba, il robot aspirapolvere. Sebbene questo tipo di robot non rappresenti un'importante innovazione dal punto di vista tecnologico e scientifico, la sua introduzione sul mercato e nelle case delle persone ha avuto un notevole impatto sociale. Roomba ha dimostrato che i robot non solo possono essere utili in ambito industriale e militare, ma anche migliorare la vita quotidiana delle persone, offrendo soluzioni pratiche e convenienti.

Dopo aver reso i robot accessibili a tutti, il passo successivo è stato migliorare la sicurezza stradale e ridurre gli incidenti causati da errori umani, rendendo il trasporto più sicuro e confortevole. Questo obiettivo è stato perseguito nel 2015 con significativi investimenti in ricerca e sviluppo da parte di aziende come Tesla e Uber. L'obiettivo è quello di costruire veicoli autonomi utilizzando sensori, intelligenza artificiale e algoritmi avanzati per guidare in modo sicuro ed efficiente senza la necessità di un conducente umano. Uno dei sistemi utilizzati per raggiungere questa guida automatica è il sistema SLAM (Simultaneous Localization and Mapping), che incrocia i dati provenienti da diversi tipi di sensori con una mappa offline, trasformandoli in una mappa

aggiornata in tempo reale. Tesla ha sviluppato il proprio sistema chiamato Autopilot, che consente la guida autonoma, ma attualmente richiede ancora la supervisione continua dell'uomo. Tuttavia, il sistema per rendere l'automobile completamente self-driving è in fase di sviluppo, con l'obiettivo di eliminare del tutto la necessità dell'intervento umano.

Guardando al futuro della robotica, è evidente che questa disciplina continuerà a trasformare il nostro modo di vivere e lavorare. L'integrazione dei robot e dell'intelligenza artificiale si espanderà, influenzando sempre più settori e incidendo profondamente sulla nostra vita quotidiana. L'innovazione in campo robotico sta aprendo nuove opportunità e presentando sfide in vari ambiti. Con l'avanzamento delle tecnologie, vedremo robot sempre più avanzati, capaci di apprendere, adattarsi e interagire con gli esseri umani in modo naturale. L'obiettivo è creare un futuro in cui i robot lavorino al nostro fianco, migliorando la qualità della vita e affrontando le sfide globali. L'impatto della robotica e dell'intelligenza artificiale è destinato a crescere, influenzando le nostre decisioni e la direzione della società nei prossimi anni.

1.3 Categorie di Robot

1.3.1 Secondo il Livello di Autonomia

Non Autonomi

I robot non autonomi sono macchine che agiscono esclusivamente sotto il comando di un operatore umano. Possono essere controllate da un software che specifica in dettaglio ogni azione da compiere, in alternativa possono essere direttamente comandati da una persona tramite un sistema di controllo remoto. Nel primo caso, un esempio tipico è costituito dai robot industriali che operano su linee di montaggio eseguendo compiti ripetitivi che sono stati programmati anticipatamente dall'uomo. Nel secondo caso, si possono considerare le macchine o i droni telecomandati.

Questi robot non sono in grado di reagire ad eventi imprevisti e necessitano istruzioni precise per ogni azione.

Semi Autonomi

I robot semi autonomi combinano il controllo umano con un certo grado di autonomia. Queste macchine possono effettuare alcune operazioni senza l'intervento umano, ma in situazioni critiche o non previste richiedono la supervisione e il controllo di un operatore.

Un esempio di robot semi autonomo è il Roomba, che è in grado di mappare e navigare autonomamente all'interno di una casa per effettuare la pulizia, ma necessita dell'intervento umano in caso di ostacoli imprevisti o se rimane incastrato. Un altro esempio può essere un drone utilizzato in un magazzino, che può navigare autonomamente per svolgere determinate operazioni ma richiede supervisione per gestire situazioni complesse.

Autonomi

I robot autonomi sono macchine capaci di percepire il contesto ambientale tramite sensori avanzati e di prendere decisioni autonome in base alla situazione in cui si trovano ad operare. Questo è possibile grazie a vari sistemi di apprendimento, tra cui quelli basati sul machine learning e sulle tecniche di intelligenza artificiale.

A differenza dei robot non autonomi, il software di un robot autonomo non agisce in maniera strettamente deterministica, ma consente alla macchina di apprendere e adattarsi continuamente utilizzando i dati ambientali rilevati dai suoi sensori. Questo permette al robot di regolare il proprio comportamento in modo sempre più raffinato e ottimizzato, migliorando la sua efficienza e la capacità di operare in ambienti complessi e dinamici.

1.3.2 Secondo le Generazioni

Nell'ambito dello sviluppo tecnologico, la robotica ha subito un'evoluzione significativa, passando da semplici strutture meccatroniche a sistemi altamente autonomi e integrati. Questa sezione esplora le diverse generazioni della robotica evidenziando le principali tecnologie e i livelli di autonomia associati a ciascuna generazione. Le informazioni presentate di seguito sono tratte da una serie di articoli e fonti autorevoli nel campo della robotica, tra cui il lavoro presentato da Haidegger (2019) intitolato Robotics 4.0 - Are we there yet?[3], un articolo di Francesco La Trofa (2022) Cos'è la robotica, come funziona e quali sono gli esempi di applicazione[7] e un altro articolo pubblicato su AI4Business da Pierluigi Sandonnini (2023) Robot, cosa sono e quali sono le previsioni future[4]. Questi studi forniscono una base solida per comprendere l'evoluzione e le caratteristiche distintive di ogni generazione della robotica.

Generazione 0

Le macchine che appartengono a questa generazione si possono considerare semplici strutture meccatroniche prive di qualsiasi grado di autonomia.

Generazione 1.0: Sistemi Preprogrammati e Teleoperati

I robot di prima generazione sono macchine non autonome, progettate per svolgere soltanto sequenze di operazioni prestabilite, preprogrammate. Questi robot operano indipendentemente dall'intervento diretto dell'uomo, seguendo istruzioni programmabili senza capacità di adattamento. Questa generazione comprende anche i sistemi teleoperati, che rappresentano i primi paradigmi di controllo nella robotica. La teleoperazione risolveva tutti i problemi cognitivi coinvolgendo l'operatore umano nel ciclo di controllo, anche in ambienti complessi come la robotica spaziale e la medicina.

Generazione 2.0: Robotica basata su Sensori

I robot di seconda generazione sono macchine autonome in grado di raccogliere dati dall'ambiente circostante grazie all'uso di sensori e prendere decisioni grazie a sistemi di apprendimento automatico. Anche se incaricati di svolgere operazioni prestabilite, questi robot possono trovare soluzioni per completare il loro compito in caso di imprevisti che alterano le condizioni dell'ambiente di riferimento. Una tecnologia abilitante chiave è stata l'integrazione di sensori forza/coppia. Questo sviluppo ha portato al riconoscimento ufficiale di una nuova categoria di robot anche nei termini di standard (ISO/TS 15066:2016 per i robot collaborativi e ISO 13482:2014 per i requisiti di sicurezza per i robot per la cura personale). La maggior parte dei robot di servizio attuali rientra in questa categoria, che richiede un'integrazione stretta tra uomo e robot.

Generazione 3.0: Robotica Automatica

I robot di terza generazione sono macchine autonome equipaggiate con sistemi avanzati di intelligenza artificiale, che permettono loro di generare autonomamente algoritmi di apprendimento automatico e di verificarne la coerenza con le operazioni da eseguire in un dato contesto ambientale. Nelle configurazioni più sofisticate, questi robot possono operare per raggiungere obiettivi specifici, indipendentemente dalle operazioni necessarie per conseguire i risultati desiderati. Questa generazione è spesso chiamata 'robotica centrata sull'uomo'e si adatta bene al concetto di sistemi cyber-fisici (CPS)[2].

Generazione 4.0: Integrazione Avanzata delle Tecnologie

La quarta generazione rappresenterà il prossimo salto rivoluzionario nell'integrazione tecnologica, sfruttando le sinergie di tutte le generazioni precedenti. Questa generazione probabilmente inizierà con le tendenze identificate

[2]https://www.internet4things.it/industry-4-0/cyber-physical-systems-cps-cosa-sono-come-stanno-rivoluzionando-il-mondo-industriale/

nell'Internet of Robotic Things (IoRT)[3] e si baserà in gran parte sull'automazione avanzata delle conoscenze cognitive, ad esempio tramite Data Science e ontologie.

1.3.3 Unmanned System

Gli Unmanned System o Vehicles (US o UV), detti anche sistemi senza pilota, sono dispositivi elettro-meccanici progettati per operare senza la presenza di un operatore umano a bordo. Gli UV possono essere controllati a distanza da un pilota remoto oppure possono navigare autonomamente grazie a percorsi e piani preprogrammati o su sistemi di automazione dinamica più complessi utilizzando sensori, intelligenza artificiale e algoritmi avanzati. Questi veicoli includono mezzi che si muovono sul terreno (Unmanned Ground Vehicles, UGV), sulla superficie del mare (Unmanned Surface Vehicle, USV), nell'acqua (Unmanned Underwater Vehicles, UUV) oppure nell'aria (Unmanned Aerial Vehicle, UAV), comunemente noti come droni. Nella figura 1.4 vengono rappresentati ti tipi di Unmanned System con le proprie applicazioni e problematiche, identificati da Eulalia Balestri nel suo studio dei sensori degli Unmanned Systems[1].

UGV (Unmanned Ground Vehicle)

I UGV sono veicoli terrestri senza pilota che vengono utilizzati in numerosi tipi di missione, come l'esplorazione spaziale, il rilevamento ambientale e le operazioni di ricerca e soccorso. Questi veicoli possono variare notevolmente in dimensioni e configurazioni in base alla missione e all'ambiente in cui devono operare, con pesi che possono andare da 500 a 25000 kg. Generalmente questi dispositivi sono equipaggiati con sensori e controlli a bordo per osservare l'ambiente e prendere decisioni autonomamente o inviare dati a un operatore umano.

[3] P. P. Ray, "Internet of Robotic Things: Concept, Technologies, and Challenges," in IEEE Access, vol. 4, pp. 9489-9500, 2016, doi: 10.1109/ACCESS.2017.2647747.

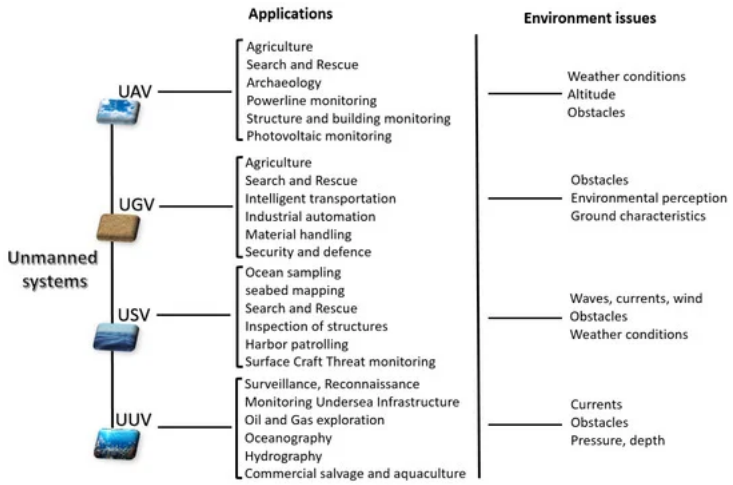


Figura 1.4: Unmanned system types and their critical environmental factors

Nonostante le capacità avanzate, gli UGV sono soggetti a limitazioni significative. Poichè i sensori a bordo non possono rilevare ostacoli oltre il loro campo visivo diretto, le condizioni ambientali come polvere, fumo, pioggia e tipo di terreno, rappresentano delle sfide significative e influenzano la velocità operativa e la capacità di evitare ostacoli del veicolo. Di conseguenza gli UGV devono essere in grado di adattare la loro traiettoria e velocità per evitare collisioni, considerando anche che la distanza di frenata aumenta con l'aumentare della velocità. Inoltre, diversi tipi di terreno, come ghiaia, fango, neve e vegetazione di varia intensità, possono complicare ulteriormente le operazioni.

La percezione dell'ambiente e il livello di autonomia dei dispositivi sono le principali caratteristiche su cui si concentrano gli sviluppi tecnologici attuali e futuri per questo tipo di veicolo senza pilota.

USV (Unmanned Surface Vehicle)

I USV sono veicoli di superficie senza pilota che operano su una superficie d'acqua e possono essere controllati a distanza oppure autonomamente. Variano in forma, peso (da decine a migliaia di kg) e velocità (da 1 m/s a 20 m/s) in base al tipo di missione specifica. Gli USV sono utilizzati in

Figura 1.5: Il Gladiator Tactical Unmanned Ground Vehicle è progettato da Emil Lien Akre nel 2005 per supportare il Corpo dei Marines degli Stati Uniti nelle missioni Ship To Object Maneuver (STOM).

condizioni pericolose per gli esseri umani, sono compatti e richiedono poca manutenzione. Inizialmente venivano impiegati per scopi navali come sorveglianza e ricognizione, oggi trovano ampio uso anche in applicazioni civili, tra cui monitoraggio ambientale, navigazione autonoma, ricerca e soccorso, rilevamento offshore, mappatura dei fondali marini e ispezione di strutture.

Figura 1.6: Sounder - Unmanned Sruface Vehicle per studi idrografici

Questi veicoli operano tra aria e acqua, richiedendo una buona percezione ambientale per evitare ostacoli sopra e sotto l'acqua e per navigare efficacemente. Condizioni meteorologiche e idriche avverse, come pioggia, vento

estremo e acqua agitata, possono influenzare significativamente le operazioni di questi dispositivi.

La ricerca si concentra sulle sfide tecnologiche legate alla durata delle missioni a lungo termine e all'operazione in condizioni estreme, nonchè sulla capacità di evitare gli ostacoli sia sopra che sotto l'acqua.

UUV (Unmanned Underwater Vehicle)

I UUV sono veicoli subacquei senza pilota che operano sotto la superficie dell'acqua con un intervento minimo o nullo da parte di un essere umano. Questi veicoli possono variare in forma, dimensione, profondità operativa e capacità di navigazione. Possono essere Remotely Operated Vehicles (ROV) controllati a distanza o Autonomous Underwater Vehicles (AUV) cioè che operano autonomamente. Gli UUV vengono utilizzati per missioni di diverso tipo, incluse sorveglianza, guerra anti-sottomarina, supporto alla costruzione e manutenzione di infrastrutture subacquee, oceanografia, idrografia e contromisure contro le mine. Gli UUV affrontano condizioni ambientali difficili come forti correnti oceaniche, alta pressione idraulica e scarsa illuminazione sott'acqua.

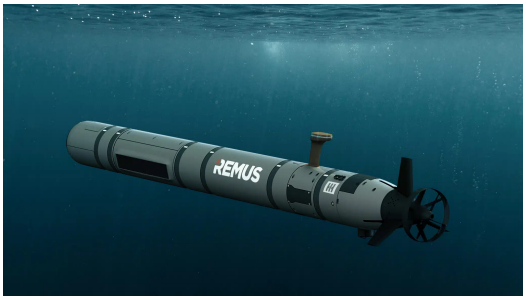


Figura 1.7: REMUS 620 - Unmanned Underwater Vehicle di classe media di HII

La navigazione, la manovrabilità e la stabilità possono essere influenzate dalle correnti oceaniche e dalla densità dell'acqua, con cambiamenti improvvisi che possono ostacolare il movimento. Inoltre, gli UUV devono rilevare e

evitare ostacoli statici o mobili e affrontare la sfida di mantenere una connessione costante con gli operatori remoti in acque profonde, rendendo cruciali le capacità di navigazione precise.

UAV (Unmanned Aerial Vehicle)

I UAV, chiamati anche droni, sono veicoli aerei senza pilota che navigano nell'aria capaci di sorvegliare ampie aree e di raggiungere ambienti ostili per l'essere umano. Possono essere controllati a distanza oppure volare autonomamente.

Esistono diversi tipi di droni, a seconda dello scopo specifico per cui sono stati progettati e la missione che devono svolgere. Gli UAV possono variare nella dimensione da pochi centimetri a decine di metri, in peso da decine di grammi a migliaia di chilogrammi, in altitudine operativa da decine di metri a trenta chilometri e in raggio d'azione da 100 metri a 1000 chilometri. La tecnologia collegata ai droni è in continuo e rapido sviluppo e il numero di applicazioni per gli UAV sta crescendo esponenzialmente, includendo il monitoraggio in tempo reale, la fornitura di copertura wireless, il telerilevamento, la ricerca e il soccorso, la consegna di pacchi, la sicurezza e la sorveglianza, l'agricoltura di precisione e l'ispezione delle infrastrutture civili.

Figura 1.8: DJI Phantom - Unmanned Aerial Vehicle per la fotografia aerea commerciale e ricreativa

L'ambiente in cui operano i UAV può influenzare la riuscita delle loro missioni. Venti estremi, pioggia e tempeste possono causare deviazioni dal

percorso prestabilito o, specialmente nel caso di droni piccoli, impedire al veicolo di operare. Un altro problema legato all'ambiente operativo degli UAV deriva dalla possibile presenza di ostacoli lungo il loro percorso, sia in ambienti esterni che interni. Gli ostacoli possono essere fermi o in movimento, rendendo più complesso evitarli. Anche l'altitudine raggiunta dai droni è un parametro importante che può essere influenzato dalle condizioni atmosferiche. Se l'elevazione dell'aria cambia rapidamente e significativamente il drone deve essere in grado di adattarsi rapidamente a questi cambiamenti raggiungendo l'altitudine richiesta. Un ulteriore limitazione da tenere a mente è il peso caricato dal drone. Il carico dei droni può influenzare le capacità di navigazione e misurazione del veicolo, nonchè la durata della missione e l'area coperta, che sono dettagli da tenere presente per la programmazione di missioni se il drone deve operare per lunghi periodi di tempo su ampie regioni di interesse.

Il peso limitato del carico utile, l'altitudine e la distanza coperta, così come l'influenza delle condizioni meteorologiche e la gestione degli ostacoli, rappresentano alcune delle debolezze degli UAV che la ricerca sta cercando di affrontare.

1.4 Software per la gestione di missioni UAV

L'utilizzo di Unmanned Aerial Vehicles (UAV), comunemente noti come droni, ha rivoluzionato molti settori, dalla sorveglianza e mappatura alla consegna di pacchi e ispezione industriale. La gestione efficace delle missioni UAV richiede software avanzati che possano pianificare, monitorare e controllare i voli dei droni in modo efficiente e sicuro. Questi software non solo migliorano la precisione e l'affidabilità delle missioni, ma permettono anche di automatizzare processi complessi e ridurre il rischio di errori umani.

Esistono diversi software per la gestione delle missioni UAV, ognuno con caratteristiche uniche e specifiche applicazioni. Di seguito, vengono presenta-

ti alcuni dei più diffusi e utilizzati, seguiti da un'analisi delle loro limitazioni in relazione alle esigenze specifiche delle missioni UAV indoor.

1.4.1 QGroundControl (QGC)

QGroundControl (QGC)[4] è una stazione di controllo a terra (Ground Control Station GCS)[5] intuitiva e potente progettata per UAV (unmanned Aerial Vehicle). Il suo obiettivo principale è la facilità d'uso sia per utenti alle prime armi che per professionisti. Fornisce controllo completo del volo e pianificazione delle missioni per qualsiasi drone abilitato MAVLink e configurazione del veicolo per UAV alimentati da PX4 e ArduPilot.

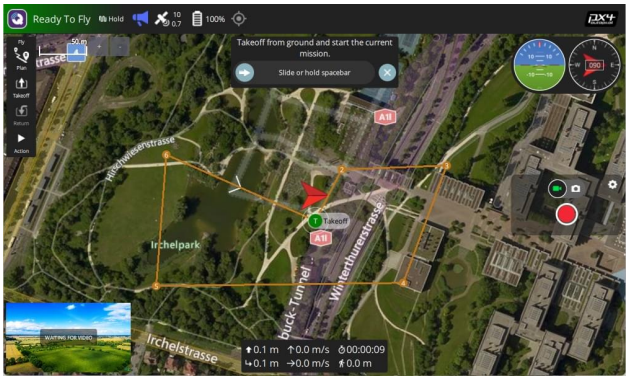


Figura 1.9: dashboard di QGroundControl

QGroundControl permette agli utenti di pianificare le missioni sulla mappa; infatti, su essa possiamo vedere la posizione del veicolo, la traccia di volo, i waypoint e gli strumenti del veicolo, fornendo una visione chiara e dettagliata delle operazioni in corso.

1.4.2 Indoor Robotics - $Tando^{TM}$

Indoor Robotics[6] offre una soluzione autonoma con il drone $Tando^{TM}$. Questo drone AI-based è progettato per monitorare e avvisare su intrusioni,

[4]http://qgroundcontrol.com/
[5]https://www.unmannedsystemstechnology.com/expo/ground-control-stations-gcs/
[6]https://www.indoor-robotics.com/

violazioni di sicurezza e potenziali pericoli. Quando non è in volo, il drone si aggancia al soffitto funzionando come telecamera di sicurezza.

Figura 1.10: The Control Bridge di Indoor Robotics per Tando

Il sistema include una piattaforma di controllo chiamata Control Bridge, che gestisce le operazioni del drone, consente la creazione di nuove missioni e programmi, e invia alert su eventi rilevati e cambiamenti rilevanti. $Tando^{TM}$ raccoglie dati visivi e termici, trasmettendoli al Control Bridge per l'analisi, che poi invia avvisi agli utenti o comandi ad altri sistemi per risolvere eventuali problemi.

1.4.3 Dragonfly

Dragonfly[7] è una tecnologia di localizzazione indoor molto precisa che utilizza solo la fotocamera a bordo del drone. Calcola la posizione del drone attraverso un flusso video, senza necessità di fusione di sensori, MEMS o hardware da installare in loco.

Dragonfly riesce a fornire una posizione molto precisa utilizzando solo una telecamera a bordo, non richiede fusione di sensori, MEMS, hardware

[7]https://dragonflycv.com/drones-uav-uas-fpv-sapr-accurate-tracking-and-autonomous-navigation-indoor/

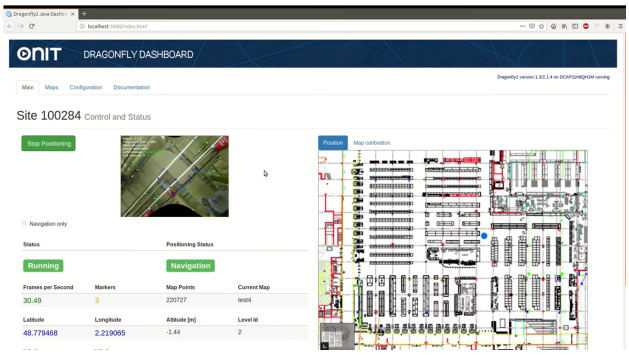


Figura 1.11: Snapshot di una missione in corso di Dragonfly

da installare sul posto o codici QR. Dragonfly necessita di un flusso video da 640 x 480 pixel dal drone. La telecamera può essere monoculare (idealmente grandangolare) o stereoscopica, e può puntare in qualsiasi direzione (preferibilmente verso il soffitto). Si può utilizzare la telecamera esistente del drone, a condizione che il flusso video possa essere consumato dal software Dragonfly.

1.4.4 TINAMU

TINAMU[8] sviluppa soluzioni per l'automazione delle ispezioni con droni. Il sistema consiste in una dashboard per la visualizzazione delle informazioni rilevanti, software di analisi su cloud sicuro e un sistema drone connesso a una rete di comunicazione.

La maggior parte degli inventari dei depositi viene effettuata manualmente su carta, generando imprecisioni, errori e rischi per la sicurezza. Poiché il processo è lungo e ripetitivo, gli inventari non vengono eseguiti con sufficiente frequenza. Questo porta a operazioni inefficienti, cattiva pianificazione, problemi nella catena di approvvigionamento, perdita di profitto e aumento delle emissioni di carbonio. Le aziende che digitalizzano i loro inventari con lidar(Light Detection and Ranging) [9], droni e altri dispositivi affronta-

[8]https://tinamu-labs.com/

[9]**lidar**: tecnologia di rilevamento remoto che utilizza impulsi laser per misurare distanze precise e creare mappe tridimensionali dell'ambiente circostante

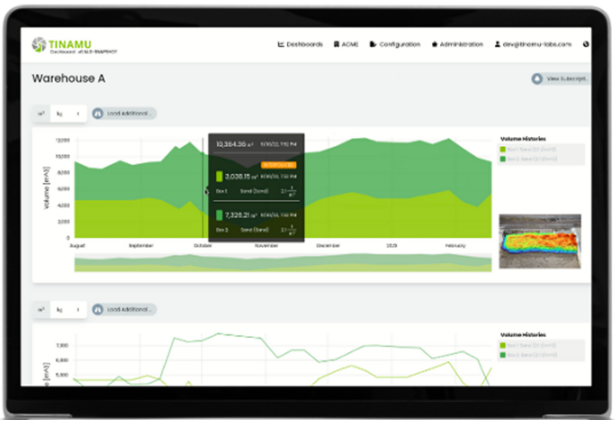


Figura 1.12: dashboard di accesso ai dati di una missione di TINAMU

no un processo manuale, lungo e oneroso. TINAMU offre una soluzione di automazione completa che permette ai responsabili della catena di approvvigionamento e degli inventari di avere fiducia, visibilità e controllo sui loro inventari nel cloud con la semplice pressione di un pulsante.

1.5 Limiti delle Soluzioni Attuali

Nonostante le diverse soluzioni disponibili esistono delle lacune nelle tecnologie attuali per la gestione delle missioni di droni in un ambiente indoor.

- QGroundControl: eccellente nella pianificazione delle missioni e nel controllo del volo dei droni, ma risulta complesso da configurare per utenti non esperti e inoltre nasce principalmente per gestire missioni outdoor, adattarlo per missioni indoor potrebbe essere complicato a causa delle specifiche esigenze della navigazione e controllo in ambienti indoor.

- Tando: ben progettato per la sicurezza e la sorveglianza di uno stabile ma non adatto per gestire missioni specifiche come la raccolta di foto. Inoltre si tratta di un drone specifico, mentre noi cerchiamo più flessibilità e versabilità e una soluzione da applicare in contesti e hardware diversi.

- Dragonfly: offre un tracking preciso ma manca la capacità di gestire missioni di droni.
- TINAMU: Ottimo per le automazioni delle ispezioni con droni ma non specifico per la gestione di missioni indoor.

Dall'analisi delle soluzioni esistenti possiamo vedere che, sebbene ci siano tecnologie avanzate per il controllo dei droni, nessuna offre una soluzione completa e adatta per la creazione di missioni di droni in un ambiente indoor con l'obiettivo di effettuare foto e misurazioni. Questo contesto evidenzia la necessità di una soluzione mirata che il nostro progetto intende soddisfare.

Capitolo 2

Background e Obiettivi

2.1 Contesto del Progetto

La tesi si focalizza sull'ideazione ed implementazione di una Web App, denominata *InDrone*, progettata per la pianificazione e la gestione delle missioni indoor con l'utilizzo di droni, in partnership con clienti del territorio emiratino.

2.1.1 Motivazioni

Come abbiamo dedotto dall'analisi dello stato dell'arte, attualmente, le soluzioni esistenti non soddisfano pienamente i requisiti richiesti dal Centro di Ricerca di Robotica, in particolare per quanto riguarda la personalizzazione delle missioni e il controllo preciso dei percorsi dei droni.

La motivazione principale di questa tesi è trovare una soluzione che fornisca agli operatori uno strumento flessibile e personalizzabile per creare e monitorare missioni specifiche, rispondendo alle esigenze emerse dalla ricerca condotta nello stato dell'arte. *InDrone* si propone di colmare questa lacuna, offrendo la possibilità di:

- **Creazione di Percorsi Personalizzabili**: Gli operatori possono scegliere il percorso del drone direttamente sulla mappa dell'edificio, defi-

nendo con precisione i nodi di un grafo.

- **Missioni Personalizzate**: L'utente può definire il tipo di missione, ad esempio scattare foto, rilevare la temperatura dell'ambiente o scattare foto termiche. Il numero di foto o i punti di rilevazione possono essere specificati direttamente dall'operatore. L'utente dovrà scegliere tra i punti del percorso quali sono i nodi obiettivo in cui effettuare le rilevazioni.

- **Monitoraggio in Tempo Reale**: Durante la missione, l'operatore può monitorare in tempo reale lo svolgimento attraverso una mappa interattiva che visualizza un anteprima del percorso che il drone andrà a svolgere, il drone in movimento e il percorso effettuato. Informazioni aggiuntive come l'altitudine, l'orientamento, la velocità del drone, l'orario di inizio e fine missione, e la durata complessiva della missione sono continuamente aggiornate.

- **Controllo della Missione**: Durante lo svolgimento di una missione, l'operatore ha il pieno controllo sul drone con la possibilià di mettere in pausa e far ripartire la missione, abortirla in modo ordinario o effettuare uno stop di emergenza. Questo garantisce un maggiore controllo e flessibilità nel caso di imprevisti o emergenze.

- **Report di Missione**: Alla conclusione della missione, viene generato un report in formato PDF con le informazioni della missione e gli obiettivi raggiunti, fornendo all'operatore una documentazione completa dell'attività svolta.

2.1.2 Obiettivi

Una volta studiato il contesto dell'applicazione sono stati definiti i seguenti obiettivi:

- **Analisi delle soluzioni presenti**: Ricerca e valutazione delle soluzioni disponibili sul mercato per la pianificazione e gestione delle missioni

con droni. Questo include lo studio delle tecnologie e degli approcci utilizzati, nonché l'identificazione dei punti di forza e delle debolezze delle soluzioni esistenti.

- **Raccolta e definizione dei requisiti**: Coinvolgimento del Autonomous Robotics Research Center (ARRC) del Technology Innovation Institute (TII) per capire le loro esigenze e aspettative specifiche. Attraverso incontri e interviste con gli esperti del centro, verranno raccolti i requisiti funzionali e non funzionali dell'applicazione, assicurandosi che risponda precisamente alle necessità operative e di ricerca del centro.

- **Progettazione dell'interfaccia grafica**: Creazione di mockup dettagliati della WebApp per valutare l'interfaccia utente. Questa fase prevede la collaborazione con il centro di robotica per ottenere feedback iterativo sui mockup, al fine di progettare un'interfaccia intuitiva e user-friendly che semplifichi l'interazione dell'operatore con l'applicazione.

- **Implementazione della soluzione proposta**: sviluppo dell'applicazione e integrazione dei vari componenti principali derivati dai requisiti.

Capitolo 3

Analisi e Progettazione

3.1 Analisi dei requisiti

Lo sviluppo del progetto inizia con un'analisi dei requisiti, che è una fase fondamentale nel processo di sviluppo software, che mira a comprendere e documentare le esigenze e le aspettative del TII, che nel mio caso è trattato come il cliente. Questo processo consente di definire in maniera chiara e precisa le funzionalità che il sistema deve offrire, oltre alle caratteristiche non funzionali come usabilità, prestazioni e manutenibilità.

3.1.1 Requisiti funzionali

Di seguito elenchiamo i requisiti funzionali che sono stati identificati dalle interazioni con il cliente.

Autenticazione Utente

L'utente deve effettuare l'autenticazione prima di poter usufruire delle funzionalità dell'applicazione. Per autenticarsi occorre inserire il proprio username e password e, se corretti, si può accedere al sistema.

Una volta che l'utente ha avviato la sua sessione deve poter eseguire il logout in modo semplice.

Creazione Missione

L'utente autenticato deve poter creare una nuova missione selezionando dei punti direttamente dalla mappa. Questi rappresentano i nodi del percorso che il drone deve effettuare. Dall'elenco dei nodi selezionati, l'utente deve sceglie i punti obiettivo, ovvero, dove il drone dovrà fermarsi per eseguire un tipo di rilevazione. Esistono diversi tipi di rilevazione possibili, ad esempio: scattare delle foto, rilevare la temperatura o scattare delle foto con la termocamera. Oltre a scegliere il tipo di obiettivo per la missione, l'operatore deve anche specificare il numero di rilevazioni da effettuare, il nome da dare alla missione e la data in cui effettuare la missione.

Selezione UAV

L'utente deve poter selezionare il drone con cui eseguire la missione. Questo potrà essere fatto attraverso la visualizzazione di una lista di droni che contiene le informazioni per ognuno. Queste ultime contengono: la marca e il modello; lo stato del drone; la sua disponibilità ad iniziale una missione; e il livello della batteria.

Storico Missioni

L'utente deve avere la possibilità di visualizzare lo storico delle missioni e poter decidere di rieseguire una missione contenuta nella lista. L'utente deve anche poter gestire questo storico attraverso la possibilità di eliminare delle missioni.

Visualizzazione della Missione sulla Mappa

Una volta fatta partire la missione, l'utente deve essere in grado di seguire lo svolgimento della missione sulla mappa in tempo reale. Viene visualizzato il percorso previsto e pianificato, il drone in movimento e il percorso effettivamente svolto dal drone.

Monitoraggio Missione

L'applicazione deve mostrare in tempo reale le informazioni relative alla missione in corso, tra cui la posizione del drone, la velocità, l'orientamento, il nome della missione e il modello del drone.

Inoltre, l'utente deve poter intervenire attivamente durante la missione tramite una serie di pulsanti che gli permettono di mettere in pausa la missione, di farla ripartire, di abortirla oppure di effettuare uno stop di emergenza.

Visualizzazione Report

Dopo il completamento della missione all'operatore deve essere fornito un report con i risultati della missione, incluse le informazioni generali della missione, i nodi del percorso del drone e le rilevazioni effettuate dal drone nei vari punti obiettivo.

3.1.2 Requisiti non funzionali

Usabilità

L'interfaccia utente deve essere intuitiva e facile da utilizzare per garantire un'esperienza ottimale. Gli operatori devono essere in grado di creare e monitorare le missioni senza la necessità di una previa formazione approfondita.

Prestazioni

L'applicazione deve essere reattiva e gestire in modo efficiente l'aggiornamento in tempo reale della posizione del drone e delle informazioni della missione in corso. Allo stesso tempo deve rispondere prontamente ai comandi dell'operatore garantendo un'esperienza fluida e senza ritardi percepibili.

Manutenibilità

Il codice deve essere modulare e ben documentato per facilitare la manutenzione e l'aggiornamento futuro dell'applicazione. Devono essere presenti commenti chiari che spiegano la logica e le funzionalità principali, rendendo il codice comprensibile e facile da modificare anche da altri sviluppatori.

3.2 Caso d'uso

Un caso d'uso è la descrizione dettagliata di un'interazione specifica tra un utente e un sistema. Questa interazione è progettata per raggiungere un obiettivo particolare dell'utente. I casi d'uso sono fondamentali per la comprensione dei requisiti funzionali di un sistema, poichè descrivono le modalità con cui gli utenti interagiranno con esso e come il sistema risponderà a queste interazioni.

Un caso d'uso ben definito include una serie di scenari che descrivono il flusso delle azioni eseguite dall'utente e dal sistema. La descrizione dettagliata dei casi d'uso aiuta a garantire che tutte le possibili interazioni siano considerate durante la fase di progettazione e sviluppo del sistema.

Diagramma dei casi d'uso

I diagrammi dei casi d'uso sono una rappresentazione grafica dei casi d'uso e dei loro attori. Essi fanno parte dei diagrammi comportamentali utilizzati nell'**Unified Modeling Language** (**UML**) per descrivere una serie di azioni che un sistema dovrebbe o può eseguire in collaborazione con uno o più utenti o attori esterni. I diagrammi dei casi d'uso sono utili per visualizzare le interazioni tra gli attori (che possono essere utenti, altri sistemi o componenti esterni) e il sistema in esame.

Attraverso il diagramma dei casi d'uso, è possibile ottenere una visione chiara e organizzata delle funzionalità del sistema e delle interazioni tra utenti e sistema stesso. Questo facilita il processo di sviluppo e contribui-

sce a garantire che il sistema finale soddisfi le esigenze degli utenti in modo completo ed efficiente.

Di seguito nella Figura 3.1 possiamo vedere il diagramma dei casi d'uso creato per questo progetto. Questo diagramma è stato creato partendo dai requisiti precedentemente identificati, infatti vengono mostrate tutte le operazione che l'utente può effettuare per interagire con il nostro sistema.

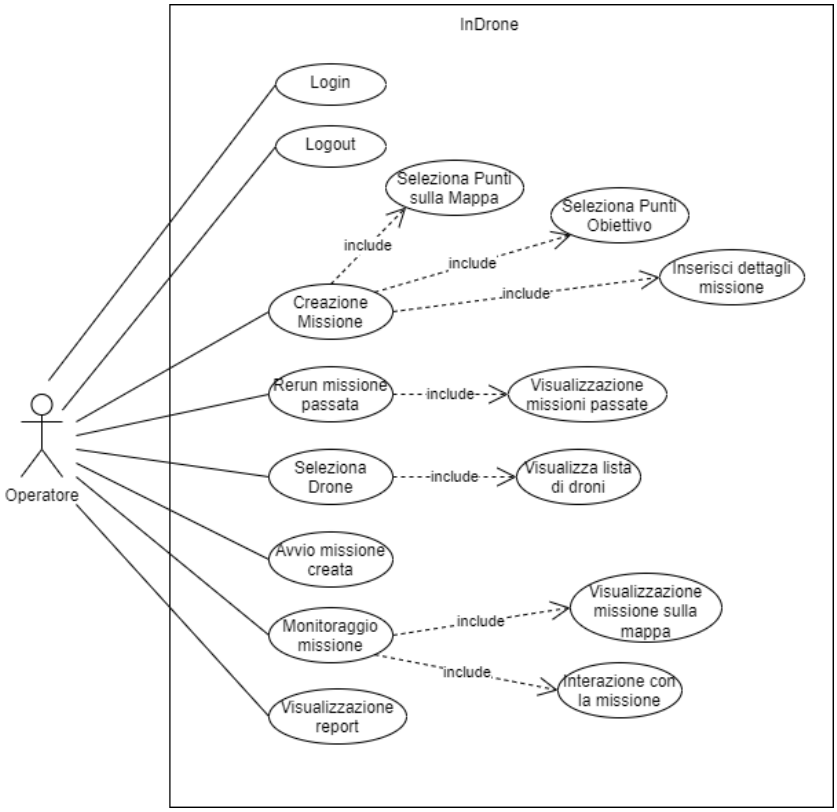


Figura 3.1: *InDrone*: diagramma dei casi d'uso

Caso d'uso: Creazione di una nuova missione e monitoraggio

Di seguito nella Tabella 3.1 descriviamo uno scenario tra tanti per spiegare in modo specifico e per dare un idea di come l'operatore può interagire con il sistema e che passi deve eseguire per farlo. In questo caso abbiamo scelto lo scenario in cui l'utente vuole creare uno nuova missione e effettuarne il suo monitoraggio.

Tabella 3.1: Caso d'uso

Attori:	Operatore
Precondizioni	L'operatore deve essere autenticato e il drone è già stato selezionato
Sequenza eventi:	1. L'operatore seleziona `+ Mission` 2. L'applicazione mostra la mappa per la selezione dei punti. 3. L'operatore seleziona i punti sulla mappa per delineare il percorso del drone. 4. L'operatore inserisce i dettagli della missione, inclusi il nome della missione, il suo obiettivo, il numero di rilevazioni, la data di esecuzione e la velocità del drone. 5. L'operatore avvia la missione. 6. L'applicazione mostra il percorso previsto sulla mappa e il percorso effettivo del drone in tempo reale. 7. L'operatore può monitorare la posizione del drone, la velocità, l'orientamento e altre informazioni in tempo reale. 8. L'operatore può mettere in pausa, riprendere, fermare o effettuare uno stop di emergenza della missione tramite i pulsanti disponibili. 9. Al termine della missione, l'applicazione genera un report e mostra un pulsante per aprirlo. 10. L'operatore clicca il pulsante `View Report`. 11. L'applicazione mostra il report all'operatore.

3.3 Architettura ad alto livello dell'Applicazione

L'architettura ad alto livello di *InDrone*è rappresentata da uno schema che raffigura come interagiscono tra di loro i vari componenti dell'applicazione. Di seguito vediamo lo schema in Figura 3.2 e una descrizione dei vari elementi che lo compongono.

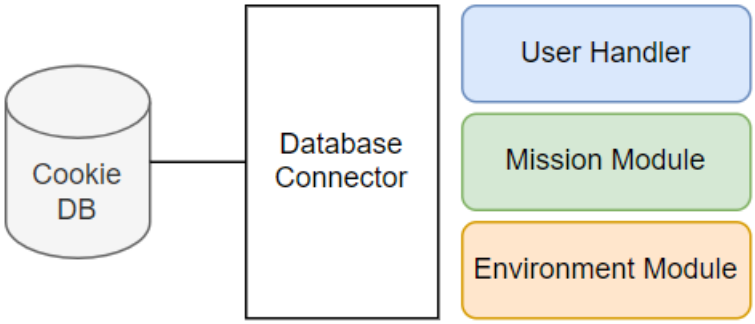


Figura 3.2: Architettura di *InDrone*

- **Cookies DB**: svolge un ruolo di mantenimento delle informazioni di sessione per gli utenti e per i droni. Vengono immagazzinate le informazioni si sessione degli utenti, come le credenziali del login, le preferenze di gestione e la selezione dei droni per le missioni, permettendo un esperienza personalizzata e persistente. I dati salvati sui cookie sono fondamentali per la gestione delle sessioni e delle preferenze utente.

- **Database Connector**: funge da intermediario tra il **Cookies DB** e i tre moduli principali dell'applicazione. Garantisce un accesso efficiente e sicuro ai dati memorizzati nei cookie, aggiornandoli e recuperandoli secondo le necessità. Il Database Connector è responsabile di rendere disponibili le informazioni di sessione agli altri moduli dell'applicazione.

- **User Handler**: è l'elemento che gestisce tutte le operazioni legate agli utenti. Questo include la gestione del login, della registrazione, il recupero delle informazioni dell'utente e la verifica che l'utente sia registrato. Lo User Handler si assicura che l'utente abbia accesso personalizzato alle funzionalità dell'applicazione.

- **Mission Module**: è il modulo che riguarda la gestione delle missioni, comprende tutte le funzioni necessarie per creare, modificare, monitorare e terminare le missioni dei droni. Inoltre, include la logica per la gestione del percorso della missione, l'assegnazione dei punti obiettivo e l'aggiornamento in tempo reale dei dati delle missioni.

- **Environment Module**: è il modulo che riguarda la gestione della mappa ed è responsabile per la presentazione grafica della mappa indoor, dell'aggiornamento della mappa in tempo reale, della raffigurazione dei percorsi dei droni e del loro aggiornamento costante e della reattività della mappa ai comandi inviati dall'utente.

L'interazione tra questi moduli è fondamentale per il funzionamento dell'applicazione. Il **Cookie DB** comunica con il **Database Connector** per fornire e aggiornare i dati dell'applicazione. A sua volta, quest'ultimo fornisce i dati ai tre moduli principali: **User Handler**, **Mission Module** ed **Environment Module**. Questo garantisce che tutte le operazioni siano basate su informazioni coerenti e aggiornate.

3.4 Interfacce Principali

Le interfacce principali di *InDrone*sono state progettate con cura e attenzione ai dettagli utilizzando **Figma**[1], un software di grafica creato appositamente per la creazione di interfacce. Figma ci ha permesso di creare

[1]**Figma**: un editor di grafica vettoriale e uno strumento di prototipazione. *https://www.figma.com/*

dei *mockup*[2] per la rappresentazione grafica della nostra applicazione. Di seguito, presentiamo le principali interfacce del nostro sistema, evidenziando le caratteristiche chiave e le scelte di design.

3.4.1 HomePage

La pagina iniziale dell'applicazione sarà proprio la **Home Page** dove verrà visualizzata la missione in corso. Da questa pagina l'utente può accedere a tutte le funzionalità dell'applicazione grazie alla presenza della **navbar**. Infatti, come possiamo vedere dalla Figura 3.3, l'utente può:

- Accedere alla schermata per il **login**;
- Accedere alle **impostazioni**;
- Accedere alla schermata + **Mission** per creare una nuova missione;

Inoltre una volta avviata la nuova missione l'utente potrà:

- Usare il pulsante più a sinistra per mettere in pausa e poi far ripartire la missione;
- Usare il pulsante in mezzo per effettuare un arresto di emergenza. In questo caso il drone atterra immediatamente senza completare la sua missione o tornare nella sua home base;
- Usare il pulsante di destra per stoppare la missione, in questo caso il drone non conclude la missione ma torna alla home base;
- Guardare la missione live nel riquadro in basso a sinistra;
- Controllare le informazione real time della missione dal riquadro in basso a destra.

[2]**mockup**: rappresentazione visiva e statica di un'interfaccia utente, solitamente utilizzata per mostrare l'aspetto e la strutture di un'applicazione

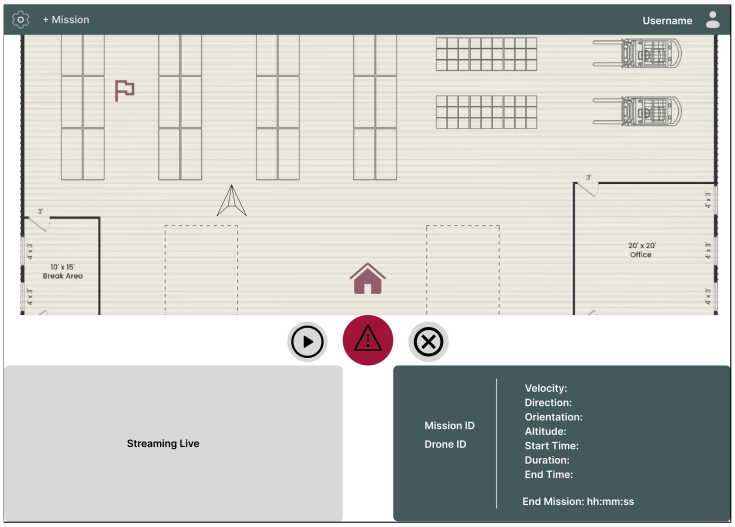


Figura 3.3: Mockup dell'Home Page di *InDrone*

3.4.2 Login

Attraverso la pagina del login, molto semplice e intuitiva, l'operatore può accedere all'applicazione inserendo le proprie credenziali negli appositi spazi denominati `Username` e `Password`. Una volta compilati i campi, può confermare il login attraverso il pulsante `Confirm`.

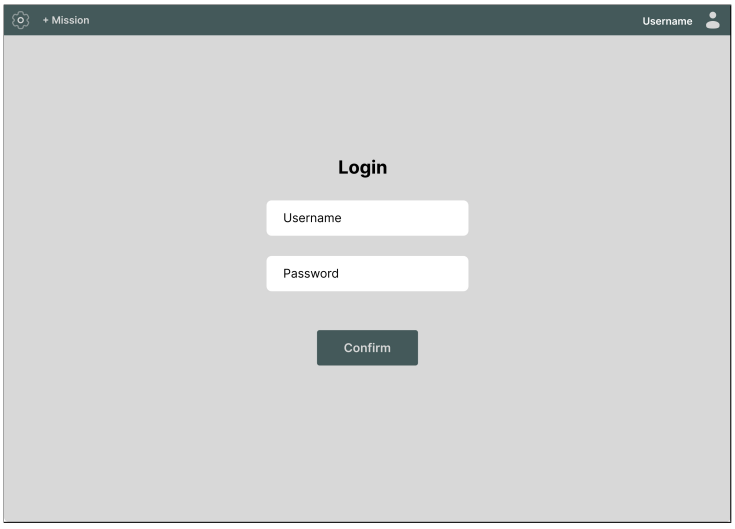


Figura 3.4: Mockup del Login di *InDrone*

3.4.3 New Mission

Dal mockup di **New Mission** (Figura 3.5) si può vedere l'importanza data alla mappa e alla selezione del percorso per il drone, infatti la mappa occupa più di metà della pagina. Cliccando su di essa compaiono dei pin che indicano: i nodi per il percorso che il drone dovrà effettuare e i nodi che rappresenteranno i punti obiettivo in cui il drone si dovrà fermare per le rilevazioni. Sotto alla mappa possiamo vedere la richiesta di diverse informazioni. Per creare una nuova missione, oltre alla selezione dei punti serve inserire:

- Un **nome** per la missione;
- Il **tipo** della missione. Il campo di fianco a Type è un menu a tendina che mostra i tipi di missione disponibili;
- La **data** in cui si vuole effettuare la missione;
- L'**orario** di partenza della missione;
- il campo **repeat** serve per indicare se la missione va ripetuta più volte.

Di fianco a queste informazioni vediamo una serie di tasti:

- **Select Drone**: porta alla pagina per selezionare con quale drone si vuole effettuare la missione;
- **Select Points**: rende la mappa cliccabile per selezionare i punti del percorso;
- **Start**: serve per far partire la missione quando programmata;
- **Start Now**: serve per far partire la missione immediatamente.

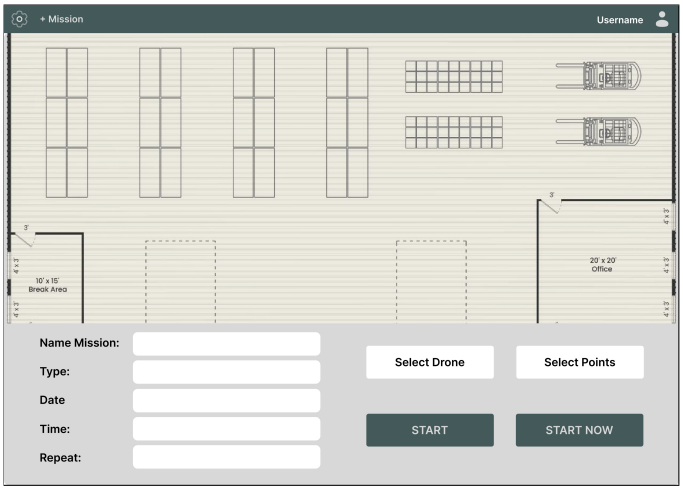


Figura 3.5: Mockup della pagina di creazione delle missioni di *InDrone*

3.4.4 Drones

Questa pagina mostra un elenco di droni tra cui poter scegliere. Per ognuno viene mostrato un riquadro che mostra le informazione personali del drone:

- **State**: lo stato del drone può avere diversi valori, tra cui: pronto, in carica, in riparazione, in viaggio o non disponibile.
- **Battery**: mostra lo stato della carica della batteria;
- **Select**: seleziona il drone per le missioni successive.

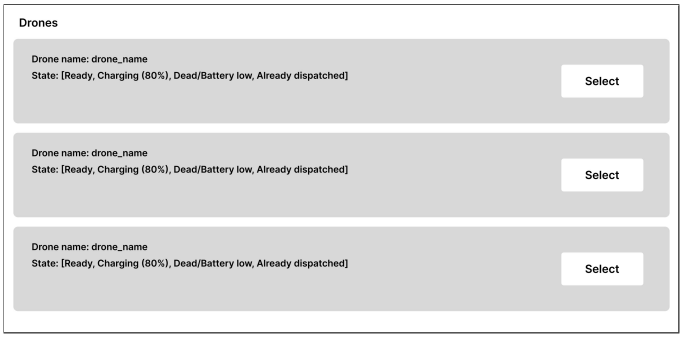


Figura 3.6: Mockup della pagine della lista di droni di *InDrone*

3.5 Problemi Principali

In questo capitolo analizziamo i problemi principali che abbiamo incontrato durante la progettazione di *InDrone*, partendo da quelli legati alla definizione e gestione del percorso.

3.5.1 Definizione del Percorso del Drone

Inizialmente, avevamo considerato l'idea permettere agli utenti di definire solamente gli obiettivi della missione cliccando direttamente sulla mappa. Idealmente, il concetto prevedeva che una volta definiti gli obiettivi, il percorso sarebbe stato calcolato automaticamente per raggiungerli.

Per facilitare il calcolo per percorso, avevamo pensato di suddividere la mappa in una griglia di quadratini, simile a quella utilizzata nel gioco "Battaglia Navale". Ogni quadratino nella griglia avrebbe rappresentato una possibile posizione del drone. Tuttavia, questo approccio presentava diversi problemi:

- **Identificazione delle Aree Percorribili**: non tutte le aree della mappa sono percorribili a causa della presenza di colonne, scaffali e altri ostacoli. Inoltre, avremmo dovuto definire manualmente quali quadratini fossero cliccabili e quali no, per ogni mappa, aumentando significativamente il lavoro di configurazione.
- **Calcolo del percorso**: per trovare un percorso possibile, avremmo dovuto utilizzare tecniche avanzate come lo sviluppo di algoritmi complessi e l'integrazione di ulteriori librerie o servizi di intelligenza artificiale.

Soluzione Finale

Alla fine abbiamo optato per un approccio più semplice e diretto: permettere agli utenti di definire manualmente il percorso del drone attraverso la definizione di nodi che, una volta connessi, formano il percorso completo.

Questo approccio ha ridotto la complessità del progetto e ha permesso una maggiore flessibilità per gli utenti nel pianificare il percorso del drone.

3.5.2 Simulazione del Percorso del Drone

Essendo il nostro progetto una simulazione e non avendo un backend, abbiamo affrontato il problema di rendere il movimento del drone il più realistico possibile. Nella realtà un drone non vola perfettamente dritto, può essere soggetto a piccole deviazioni e oscillazioni. Per simulare questo comportamento abbiamo adottato un approccio che prevedeva:

- **Segmentazione del Percorso**: il percorso del drone viene suddiviso in piccoli segmenti e ogni segmento rappresenta un piccolo tratto del percorso complessivo del drone.
- **Deviazioni Casuali**: a ciascun segmento è stata applicata una piccola deviazione casuale per simulare le oscillazioni del drone durante il suo voto. Questo a reso il movimento del drone più realistico e meno prevedibile.

3.5.3 Conclusioni

Il processo di definizione e simulazione del percorso del drone ha coinvolto varie sfide, dalla pianificazione del percorso alla simulazione del volo realistico. Optare per una definizione manuale del percorso ha semplificato il progetto ed effettuare una segmentazione del percorso in piccoli segmenti ha permesso di simulare il volo di un drone in modo più realistico. Questi approcci hanno permesso di creare una soluzione funzionale e adattabile alle esigenze degli utenti.

Capitolo 4

Implementazione

In questo capitolo analizziamo come sono state composte e sviluppate le funzionalità essenziali di *InDrone*: la creazione di una missione, la visualizzazione e gestione di una missione in corso e il report finale.

4.1 Progettazione Componenti e Servizi

Di seguito elenchiamo i componenti e i servizi principali che verranno implementati in questa applicazione. Inoltre analizziamo attraverso uno schema come questi elementi interagiscono tra di loro (Figura 4.1).

Componenti principali

I componenti principali dell'applicazione sono: login, navbar, map, new-mission, mission-list, buttons, info-mission e report. Ogni componente ha il suo scopo principale e di seguito li analizziamo brevemente:

- **login**: permette all'utente di effettuare il login nell'applicazione;
- **navbar**: si occupa della navigazione, infatti attraverso la navbar possiamo:
  - accedere alla pagina del login;

  - una volta effettuato l'accesso, possiamo vedere l'username dell'operatore ed effettuare il logout;
  - accedere alla pagine per creare una nuova missione;
  - tornare nella home page;
  - accedere alla pagina delle impostazioni.

- **map**: si occupa di visualizzare la missione in corso e d è il componente centrale della nostra applicazione;
- **new-mission**: adibito alla creazione di una nuova missione e contiene anche il componente mission-list;
- **mission-list**: visualizza le missione passate e permette all'operatore di rieseguirle.
- **buttons**: mostra e gestisce i pulsanti attraverso i quali l'operatore interagisce con la missione in corso;
- **info-mission**: mostra le informazioni in tempo reale della missione in corso;
- **report**: genera e mostra il report alla fine della missione.

Servizi

L'applicazione utilizza diversi servizi per gestire le varie funzionalità. Di seguito è riportato un elenco dei servizi e una breve descrizione di ciascuno.

- **AuthService**: Gestisce l'autenticazione dell'utente. Si occupa di login, logout, verifica se un utente è autenticato e fornisce l'username dell'utente attualmente in sessione.
- **CoordinatesService**: Gestisce le coordinate utilizzate nell'applicazione per creare una nuova missione. Consente di aggiungere, aggiornare, rimuovere e cambiare il ruolo delle coordinate. Inoltre, gestisce i marker

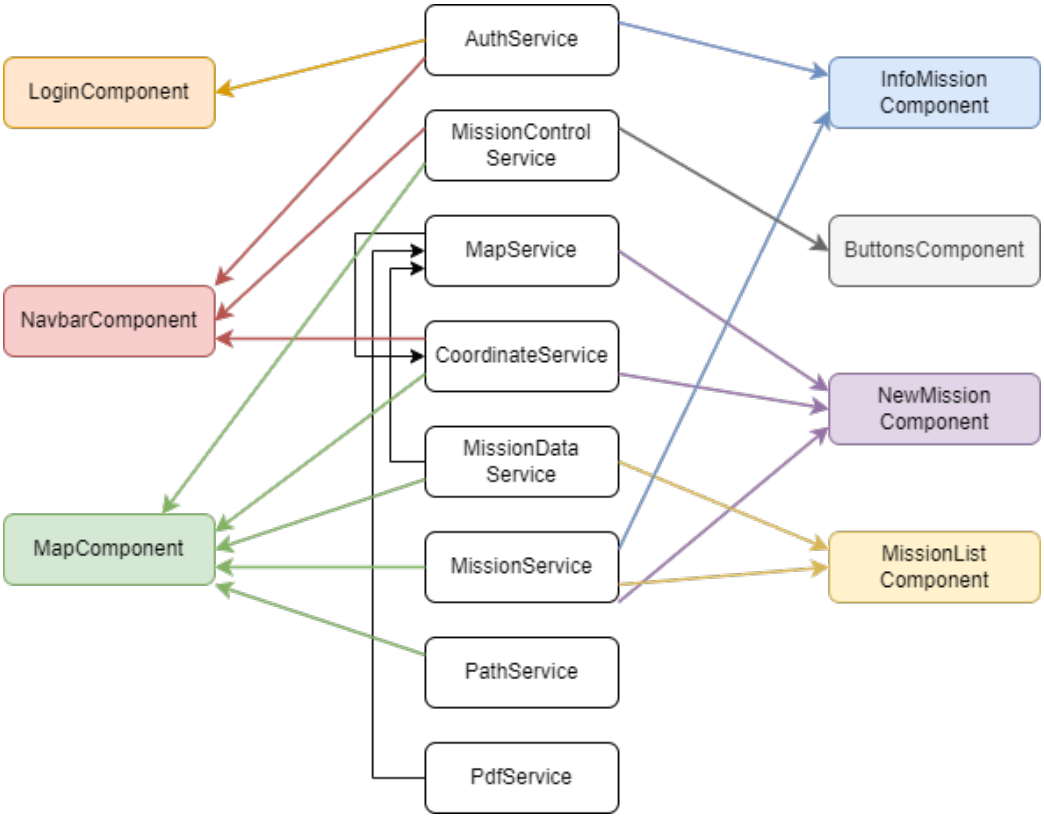


Figura 4.1: Diagramma delle interazioni servizi-componenti

sulla mappa associati a ciascuna coordinata, gestendo anche il colore del marker, associato al ruolo.

- **MapService**: Gestisce la configurazione della mappa, l'impostazione della base di partenza del drone (home base) e l'interazione con la mappa di Leaflet.
- **MissionControlService**: Gestisce il controllo delle missioni in corso. Questo include la pausa, la ripresa e l'interruzione delle missioni.
- **MissionDataService**: Gestisce l'archiviazione, il recupero e la cancellazione delle missione passate.
- **MissionService**: Gestisce il ciclo di vita delle missioni, inclusi la creazione, l'aggiornamento, il completamento e il riavvio di una missione.

- **PathService**: Gestisce il percorso che il drone deve seguire durante una missione. Questo include la definizione dei punti di inizio e fine, nonché il calcolo del percorso effettivamente seguito dal drone.

- **PdfService**: Gestisce la creazione e formattazione del pdf del report di una missione.

4.2 Tecnologie e Strumenti Utilizzati

Per lo sviluppo dell'applicazione, è stato utilizzato Angular 17. Per la gestione delle mappe è stata scelta la libreria Leaflet (versione 1.9.4) integrata con il plugin leaflet.motion (0.3.2) per effettuare i movimenti del drone all'interno della mappa. La generazione del PDF è stata gestita dalla libreria jsPDF (2.5.1.). Infine per i calcoli riguardandi il direzionamento del drone è stata usata Turf.js (5.1.5.).

Di seguito verranno spiegati tutti questi strumenti.

4.2.1 Framework Angular

Angular è un framework open-source sviluppato da Google per la realizzazione front-end di applicazione web dinamiche e complesse. Il suo scopo principale è semplificare lo sviluppo di applicazioni web, fornendo una struttura di base e una serie di strumenti che facilitano la costruzione di interfacce utente dinamiche e reattive. Per creare queste applicazioni Angular utilizza HTML/CSS e TypeScript[1]. Angular fornisce agli sviluppatori una potente piattaforma per creare applicazioni web sofisticate in modo efficiente, permettendo di importare librerie TypeScript per implementare funzionalità all'interno delle applicazioni in modo veloce e sicuro.

[1]**Typescript**: un linguaggio di programmazione basato su JavaScript che consente di scrivere codice più robusto e sicuro, riducendo errori e semplificando il debugging

Angular CLI e npm

Per creare un nuovo progetto Angular è molto importante la creazione di un ambiente di sviluppo adatto, considerando la configurazione e integrazione di pacchetti software che potrebbero cambiare nel tempo o che vorremmo sostituire. Per semplificare questo compito, il team di Angular ha messo a disposizione **Angular CLI**[2], un'interfaccia a riga di comando per creare la struttura già configurata. La configurazione di Angular CLI richiede l'utilizzo di **npm**, un package manager di JavaScript basato su Node JS[3] per l'installazione e configurazione di software.

Struttura del Progetto Generato da Angular CLI

Angular CLI genera una struttura predefinita che include vari file e directory essenziali per lo sviluppo dell'applicazione. Questa struttura ci permette di capire l'importanza dei componenti e servizi in Angular, che andremo a spiegare di seguito.

Component

I componenti possono considerarsi i blocchi fondamentali con cui si costruisce un'applicazione Angular. Questo permette di avere una struttura organizzata in parti con compiti precisi, bene organizzate e facilmente gestibili. Possiamo creare un nuovo `component` utilizzando direttamente i comandi di AngularCLI, che ci forniscono tutti i file che costituisco un componente, come possiamo vedere dalla Figura 4.2, senza doverci preoccupare di farlo noi.

Di seguito indichiamo i ruoli di ogni file che forma il componente:

- **home.component.ts** è il file TypeScript del componente;
- **home.component.html** è il file che contiene template HTML del componente;

[2]**AngularCLI**: https://angular.dev/tools/cli

[3]**Node.js**: ambiente di runtime JavaScript ottimizzato per la scalabilità di applicazioni server-side asincrone e ad alte prestazioni. https://nodejs.org/en

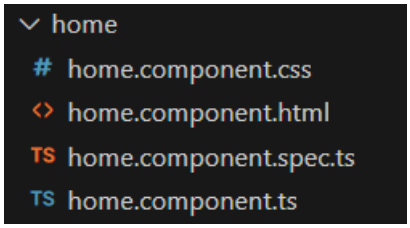


Figura 4.2: File che compongono un componente Angular

- **home.component.css** è il file per gli stili CSS del componente;
- **home.component.spec.ts** è il file per i test unitari per il componente.

Un aspetto essenziale e che viene ampiamente utilizzato nelle applicazione Angular è il **Data binding**.

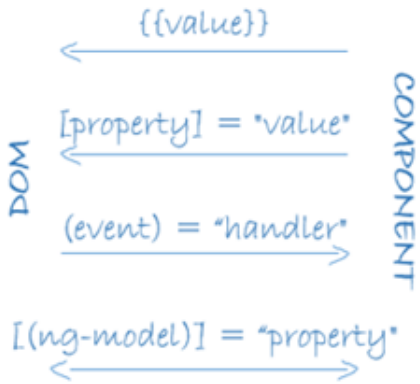


Figura 4.3: Diagramma dei modi per effettuare il two-way data binging[5]

Angular supporta questo meccanismo di **two-way data binding** che permette di sincronizzare i dati tra la parte logica (TypeScript) e la parte visiva (HTML) di un applicazione, quindi mette in comunicazione i componenti con i loro template. Il diagramma rappresentato il figura 4.3 rappresenta quattro modi in cui attraverso il template HTML possiamo connettere le due parti.

Servizi

In Angular, un servizio è una classe che fornisce un valore, una funzione o una funzionalità di cui un'applicazione ha bisogno. Un servizio ha tipica-

mente uno scopo ben definito e ristretto. Angular distingue chiaramente tra componenti e servizi per aumentare la modularità e la riusabilità del codice.

La parte di Angular che fornisce ai componenti l'accesso ai servizi viene chiamata **Dependency injection(DI)** o iniezione delle dipendenze. Per poter iniettare un servizio all'interno di un componente come dipendenza occorre utilizzare il decoratore `@injectable`. Dalla parte nel componente, dopo aver importato il servizio occorre crearne un istanza nel costruttore del componente, dopo di che si può usufruire del servizio.

Figura 4.4: Dependency Injection(DI) [6]

4.2.2 Leaflet

Leaflet[4] è una libreria di JavaScript open-source per la creazione di mappe interattive. Creata nel 2011 da Volodymyr Agafonkin, uno sviluppatore ucraino, Leaflet è oggi conosciuta per la sua leggerezza, semplicità d' uso e versatilità, offre numerose funzionalità, tra cui il supporto per la visualizzazione di marker, pop-up, layer e la gestione di eventi. La sua struttura modulare e l'ampia disponibilità di plugin la rendono una scelta ideale per progetti che richiedono mappe personalizzabili e dinamiche.

In *InDrone*, Leaflet è stata utilizzata per visualizzare la mappa dell'edificio, mostrare la posizione della home base del drone, mostrare il drone in movimento e i percorsi ideali ed effettivi dei droni.

L'integrazione di Leaflet con Angular è stata resa possibile grazie all'installazione di librerie apposite che facilitano l'uso dei componenti di Leaflet all'interno dell'applicazione Angular.

[4]**Leaflet**: https://leafletjs.com/

Dopo aver installato il pacchetto `leaflet` all'interno del progetto, utilizzando sempre `npm`, possiamo proseguire con la creazione di una mappa e dei marker all'interno di essa.

leaflet.motion

Leaflet.Motion[5] è un plugin per la libreria Leaflet che consente di animare marker e polyline[6] su una mappa, creando effetti di movimento fluidi e realistici.

Nel contesto del nostro progetto di gestione di missioni dei droni all'interno di un magazzino, leaflet.motion ci è stato particolarmente utile per visualizzare in tempo reale il movimento del drone e del rispettivo percorso sulla mappa. Grazie a questo plugin, l'operatore può vedere il drone eseguire il percorso stabilito, con aggiornamenti fluidi e intuitivi.

4.2.3 Altre Tecnologie

jsPDF

jsPDF[7] è una libreria open source JavaScript per creare file PDF in modo dinamico. jsPDF fornisce una varietà di funzionalità, tra cui l'aggiunta di testo, immagini, grafici, tabelle e forme geometriche, rendendola una soluzione versatile per molte esigenze di generazione di PDF.

In *InDrone*, jsPDF è stato utilizzato per creare il pdf del report perchè fornisce tutti gli strumenti di cui abbiamo bisogno, cioè l'inserimento di testo per elencare le coordinate dei punti in cui il drone si è fermato per eseguire delle rilevazioni; l'inserimento di immagini per inserire le foto scattate dal drone nel caso l'obiettivo della missione fosse scattare foto o foto termiche.

[5]**leaflet.motion**: https://github.com/Igor-Vladyka/leaflet.motion

[6]**Polyline**: una sequenza di linee unite definite da un elenco di punti, nel nostro caso coordinate sulla mappa

[7]**jsPDF**: https://rawgit.com/MrRio/jsPDF/master/docs/

Turf.js

Turf.js[8] è una libreria open source JavaScript che fornisce strumenti per l'analisi spaziale e la manipolazione dei dati geografici. Questa libreria è progettata per essere leggera e modulare e offre una vasta di funzionalità per la gestione di operazioni geospaziali complesse, come la misurazione di distanze, il calcolo di aree, la gestione di geometrie e l'esecuzione di operazioni di trasformazione.

In questo progetto Turf.js è stato utilizzato per la sua funzione di bearing che consente di calcolare l'angolo di orientamento tra due punti geografici. Questo ci consente di ottenere l'orientamento del drone durante una missione e di averne una rappresentazione precisa sulla mappa.

4.3 Sviluppo dei Componenti Principali

In questa sezione verrà spiegato lo sviluppo dei due componenti principali di *InDrone*, il componente per la creazione di nuove missioni **NewMissioneComponent** e quello per il monitoraggio delle missioni in corso **MapComponent**.

4.3.1 NewMissionComponent

Questo è un componente importante per l'applicazione perchè si occupa di tutto ciò che riguarda la creazione e l'avvio di una nuova missione. **NewMissionComponent** consente agli utenti autenticati di definire il percorso del drone, definire quali coordinate sono i punti obiettivo, selezionare il tipo di rilevazioni che il drone dovrà effettuare e avviare la missione.

new-mission.component.ts

Questo file TypeScript contiene la logica del componente NewMission. Dal decoratore `@Component` possiamo vedere che il componente è formato

[8] **Turf**: https://turfjs.org/

da: il selettore `app-new-mission`, i moduli importati, il template HTML `new-mission.component.html` e il file di stile `new-mission.component.css`. Inoltre analizzando il costruttore è possibile identificare tutti i servizi che sono stati utilizzati all'interno di questo componente.

```
@Component({
  selector: 'app-new-mission',
  standalone: true,
  imports: [CommonModule, FormsModule, NavbarComponent, RouterModule,
      ReactiveFormsModule, MatIconModule, MissionsListComponent],
  templateUrl: './new-mission.component.html',
  styleUrl: './new-mission.component.css'
})
export class NewMissionComponent {
  private map!: L.Map;
  missionForm!: FormGroup;
  editFlags: boolean[] = [];
  editRole: boolean[] = [];
  private dronePath?: L.Layer;
  showScrollBtn: boolean = false;

  constructor(private missionService: MissionService,
    private mapService: MapService,
    private formBuilder: FormBuilder,
    private coordinatesService: CoordinatesService,
    private route: ActivatedRoute,
    private router: Router,
    private dialog: MatDialog
  ) { }
```

Di seguito descriviamo i compiti principali di cui si occupa **NewMissionComponent**.

- **Gestione della mappa**: uno del compiti principali del componente, infatti, attraverso le iterazioni con la mappa, l'operatore seleziona i nodi per formare il percorso che il drone deve seguire.

  L'inizializzazione della mappa avviene all'interno di `ngOnInit` con la chiamata del **MapService**.

  ```
  ngOnInit() {
    this.map = this.mapService.getMap();
    this.map.panTo(new L.LatLng(240, 375))
    this.initializeForm();
    this.setupMapClickListener();
  ```

```
  this.subscribeToCoordinateChanges();

  // Get the mission id from the route
  this.route.params.subscribe(params => {
    const missionId = params['id'];
    if (missionId) {
      this.loadMission(missionId);
    }
  });
}
```

Dopo che la mappa è stata creata viene chiamato `setupClickListener()` che, utilizzando il metodo `addMapClickListener()` rende la mappa cliccabile e una volta cliccata restituisce le coordinate del punto. Queste coordinate vengono salvate e, attraverso il metodo `addCoord()` di **CoordinateService**, vengono aggiunte alla lista osservabile di nodi che compongono il percorso del drone.

```
setupMapClickListener() {
  this.mapService.addMapClickListener((e: L.LeafletMouseEvent) => {
    const latlng = e.latlng;
    this.coordinatesService.addCoord({ lat: latlng.lat, lng: latlng.
        lng, role: CoordinateRole.Point }, this.map);
    console.log(latlng);
  });
}
```

- **Gestione Form della Missione**: viene utilizzato `ReactiveFormModule` per gestire un form reattivo che consenta agli utenti di specificare i dettagli della missione. Il form viene inizializzato con un gruppo di controlli, incluso un array di coordinate che verrà aggiornato mano a mano che vengono aggiunti nodi al percorso. Il **CoordinatedService** viene utilizzato per gestire le coordinate iniziali e settare la home base come punti di partenza e fine del percorso.

```
private initializeForm() {
  this.missionForm = this.formBuilder.group({
    name: [''],
    type: [''], // camera, temperature, thermal-camera
    numberOfReadings: [1],
    waitingTime: [],
```

```
    description: [''],
    startDate: [new Date().toISOString().substring(0, 10)],
    endDate: undefined,
    droneSpeed: [1],
    coordinates: this.formBuilder.array([])
  });

  this.coordinatesService.clearCoords();
  this.coordinatesService.setHomeBase();
  this.coordinatesService.getCoords().subscribe(coords => {
    this.updateCoordinatesForm(coords);
  });
}
```

- **Gestione delle Coordinate**: utilizzando il metodo in `intializeForm()` `coordinatesService.getCoords()`, visto nel codice sopra, ci viene restituito l'Observable delle coordinate e con il metodo `.subscribe` ci sottoscriviamo per ascoltare questo Observable. In questo modo, quando il valore viene aggiornato, e quindi l'Observable emette un valore, viene chiamata la funzione di callback, che nel nostro caso è `updateCoordinatesForm()` che aggiunge una nuova coordinata al form.

```
// update the form array with the new coordinates
updateCoordinatesForm(coords: Coordinate[]) {
  this.coordinates.clear(); // clear the form array
  coords.forEach(coord => {
    if (coord && coord.lat !== undefined && coord.lng !== undefined) {
      this.coordinates.push(this.formBuilder.group({
        lat: coord.lat,
        lng: coord.lng,
        role: coord.role
      }));
    }
  });
}
```

- **Modifica delle Coordinate**: una volta inserite le coordinate, queste possono essere modificate in diversi modi:
  - **Modifica**: per modificare le coordinate vengono utilizzati diversi metodi, `toggleEdit()` per rendere possibile la modifica,

`saveChanges()` per salvare le modifiche fatte, aggiornando le coordinate e i marker associati, e `cancelChanges()` per annullare la modifica. Tutto questo viene fatto sempre grazie all'ausilio del **CoordinatesService**.

```
toggleEdit(index: number) {
  this.editFlags[index] = !this.editFlags[index];
}
// Saves changes made to a coordinate and updates the map markers
saveChanges(index: number) {
  const coord = this.coordinates.at(index).value;
  this.coordinatesService.updateCoord(index, coord, this.map);
  this.coordinatesService.updateAllMarkers(this.map);
  this.editFlags[index] = false;
}
cancelChanges(index: number) {
  // reset the form control to the original value
  const coord = this.coordinatesService.getCoord(index);
  this.coordinates.at(index).patchValue(coord);
  this.editFlags[index] = false;
}
```

– **Rimozione**: per rimuovere una coordinata, dobbiamo rimuovere la coordinata dall'Observable attraverso l'uso del **CoordinateService**. Inoltre occorre rimuovere anche i diversi tipi di `flag`, utilizzate per tenere traccia del ruolo e della modifica della coordinata.

```
removeCoordinate(index: number) {
  this.coordinates.removeAt(index);
  this.coordinatesService.removeCoord(index, this.map);
  this.editFlags.splice(index, 1);
  this.editRole.splice(index, 1);
}
```

– **Modifica il ruolo**: Una volta definite le coordinate, l'utente può modificarne il ruolo per decidere in quali coordinate il drone deve effettuare le rilevazioni, attraverso l'utilizzo del metodo `toggleRole()` che invoca il **CoordinatesService** e modifica il ruolo della coordinata scelta. In questo modo quel nodo del percorso diventa un punto obiettivo.

```
toggleRole(index: number) {
  this.coordinatesService.changeRole(index, this.map);
  console.log('Coords from new-mission after role change', this.
      coordinatesService.getCoords());
  this.editRole[index] = !this.editRole[index];
}
```

- **Gestione delle Missione Passata**: per caricare tutte le missioni passate viene utilizzato il metodo `loadCompletedMissions()` che utilizza **MissionDataService** per ottenere l'elenco delle missioni. Una volta che abbiamo le missioni caricate possiamo decidere di fare un **Rerun** della missione oppure **Delete** della missione dallo storico. Per cancellarla viene usato lo stesso servizio che, attraverso `deleteMission`, elimina la missione e poi richiama `loadCompletedMissions()` per aggiornare lo storico. Per effettuare il Rerun viene chiamato il metodo `rerunMission()` che a sua volta chiama l'omonimo metodo del **MissionService** che si occupa di caricare i dati direttamente nel form, così da essere modificabili oppure pronti per far ripartire la missione.

```
loadCompletedMissions() {
  this.completedMissions = this.missionDataService.getMissions();
}
rerunMission(missionId: string): void {
  this.missionService.rerunMission(missionId);
}
deleteMission(missionId: string): void {
  this.missionDataService.deleteMission(missionId);
  this.loadCompletedMissions();
}
```

- **Visualizzazione dei Dati**: Per visualizzare i dati relativi alla missione che si sta creando, incluso il percorso del drone sulla mappa, viene usato `subscribeToCoordinateChanges()`. Questo metodo permette l'aggiornamento del percorso quando cambiano le coordinate. Ogni cambiamento genera un aggiornamento della polyline che rappresenta il percorso, come spiegato nella sezione 4.2.2 dedicata a leaflet.motion.

```
// Listens for changes in the coordinates array and updates the drone
    path on the map
subscribeToCoordinateChanges() {this.coordinatesService.getCoords().
    subscribe(coords => {
  // delete the old drone path
  if (this.dronePath) {
    this.map.removeLayer(this.dronePath);
  }
  // filter the non valid coordinates before creating the polyline
    const validCoords = coords.filter(coord => coord && coord.lat !==
        undefined && coord.lng !== undefined);

    if (validCoords.length > 0) {
      this.dronePath = L.polyline(validCoords.map(coord => [coord.lat,
          coord.lng]), {
        color: 'red'
      }).addTo(this.map);
    }
  })
}
```

- **Avvio della Missione**: attraverso il metodo `startMisiso()` verifichiamo che form contenga dati validi e che sia completato. Dopo di che, utilizzando il **MissionService** facciamo partire la missione e navighiamo verso la home page per vederla in azione.

```
startMission() {
  if (this.missionForm.valid) {
    const formValue = this.missionForm.value;
    const missionData: Mission = {
      ...formValue,
      coordinates: this.coordinatesService.getCoordsSnapshot()
    };
    this.missionService.createMission(missionData);
    this.router.navigate(['']);
  }
}
```

new-mission.component.html

Questo file HTML rappresenta il template del componente **NewMission** e definisce l'interfaccia utente per creare le missioni dei droni. Questo template incorpora vari elementi e direttive di Angular per fornire un interfaccia

interattiva e facile da usare. Di seguito analizzeremo i vari elementi di questo template.

- **Contenitore della Mappa**: La mappa iterativa viene visualizzata all'interno di un contenitore div con `id="map"`. A questo contenitore è associato del codice CSS per visualizzare la mappa in modo appropriato.

- **Form della Nuova Missione**: La parte centrale del tamplate è occupata dal form che permette agli utenti di visualizzare e modificare le coordinate inserite e compilare i dettagli della missione. Questo form è reattivo e utilizza il binding 'formGroup' per legare i controlli del form al modello di dati gestito dal componente TypeScript.

  - **Coordinate**: la visualizzazione delle coordinate è realizzata utilizzando la direttiva Angular `*ngFor` che itera sull'array `coordinates` mostrando le loro latitudini e longitudini. Gli utenti possono modificare le coordinate cliccando sul pulsante di modifica (icona a forma di matita), che abilita i campi di input per permettere le modifiche; questa funzionalità è gestita tramite la direttiva `*ngIf`. Quando la modalità di modifica è attiva, viene utilizzato `ng-template #editMode` per rendere i capi della latitudine e longitudine modificabili e aggiunge due pulsanti per salvare o annullate le modifiche. Inoltre, il pulsante con l'icona di bandiera o puntina permette di cambiare il ruolo della coordinata (goal o punto intermedio), mentre il pulsante di eliminazione (icona a forma di cestino) consente di rimuovere la coordinata dalla lista.

```
<div class="coordinates" *ngFor="let coord of coordinates.
    controls; let i = index" [formGroupName]="i">
  <div class="coordinate">
    <label for="lat">{{i + 1}}:</label>
      <div *ngIf="!editFlags[i]; else editMode">
        <label for="coord" id="lat-lng">Lat: {{coord.value.lat}}
            - Lng: {{coord.value.lng}}</label>
          <button id="edit-btn" type="button" (click)="toggleEdit
              (i)">
```

```
            <mat-icon>
              <span class="material-icons-round"> edit </span>
            </mat-icon>
          </button>
        </div>
        <ng-template #editMode>
          <div class="edit-mode-container">
            <input formControlName="lat" placeholder="Latitudine"
                 [disabled]="!editFlags">
            <input formControlName="lng" placeholder="Longitudine
                " [disabled]="!editFlags">

            <button type="button" (click)="saveChanges(i)" *ngIf=
                "editFlags[i]">
              <mat-icon>
                <span class="material-icons-round"> done </span>
              </mat-icon>
            </button>
            <button type="button" (click)="cancelChanges(i)" *
                ngIf="editFlags[i]">
              <mat-icon>
                <span class="material-icons-round"> close </span>
              </mat-icon>
            </button>
          </div>
        </ng-template>
        <button type="button" (click)="toggleRole(i)">
          <mat-icon>
            <span class="material-icons-round">
              {{ coord.value.role === 'goal' ? 'flag' : '
                  location_on' }}
            </span>
          </mat-icon>
        </button>
        <button id="delete-btn" type="button" (click)="
            removeCoordinate(i)">
          <mat-icon>
            <span class="material-icons-round"> delete </span>
          </mat-icon>
        </button>
      </div>
```

– **Dettagli missione**: il componente gestisce la definizione dei dettagli della missione come il nome, il tipo di missione, il numero di letture da effettuare, la data di inizio e la velocità del drone. Que-

sti dettagli sono raccolti in un form in cui i dati vengono inseriti tramite campi input. Il tipo di missione viene selezionato da un menu a tendina che utilizza un `*ngFor` per elencare tutti i tipi di missione. Inoltre l'interfaccia include un pulsante che permette di navigare verso la pagina per selezionare il drone con cui effettuare la missione.

```
<div class="form-input">
  <div class="form-input-or">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" formControlName="
        name">
  </div>
  <div class="form-input-or">
    <label for="type">Type:</label>
    <!-- <input type="text" id="type" name="type" formControlName
        ="type"> -->
    <select id="type" name="type" formControlName="type">
      <option *ngFor="let type of missionTypes" [value]="type.
          name">{{ type.name }}</option>
    </select>
  </div>
  <div class="form-input-or">
    <label for="numberOfReadings">Number of readings:</label>
    <input formControlName="numberOfReadings" type="number"
        placeholder="Readings" id="numberOfReadings"
          name="numberOfReadings">
  </div>
  <div class="form-input-or">
    <label for="startDate">Date:</label>
    <input type="date" id="startDate" name="startDate"
        formControlName="startDate">
  </div>
</div>
<div class="form-input drone">
  <div class="form-input-or">
    <label for="droneSpeed">Drone speed:</label>
    <input type="number" id="droneSpeed" name="droneSpeed"
        formControlName="droneSpeed">
    <label for="unit">m/s</label>
  </div>
  <div class="form-input-or">
    <!-- <button class="drone-choice" routerLink="/drones-list">
        Select Drone</button> -->
    <button class="drone-choice" (click)="openAlert()">Select
```

```
        Drone</button>
  </div>
</div>
```

- **Lista delle Missioni Passate**: la lista delle missioni passate viene visualizzata utilizzando il componente **MissionsListComponent**. Tramite l'uso di `*ngIf` viene controllato che l'array di missioni non sia vuoto, se risulta vuoto viene mostrato il template con riferimento `#noMissions`. Utilizzando `*ngFor` vengono mostrate tutte le missioni con le loro informazioni e due pulsanti per fare il **Rerun** o il **Delete**.

```
<div class="missions-container">
  <div *ngIf="completedMissions.length > 0; else noMissions">
    <h2>Completed Missions</h2>
    <ul>
      <li *ngFor="let mission of completedMissions">
        {{ mission.name }} - {{ mission.type }} - {{ mission.startDate
            | date }} - {{ mission.endDate | date }}
        <button *ngIf="mission.id" (click)="rerunMission(mission.id)">
            Rerun</button>
        <button *ngIf="mission.id" (click)="deleteMission(mission.id)"
            >Delete</button>
        <ul>
          <li *ngFor="let coord of mission.coordinates">
            Lat: {{ coord.lat }} - Lng: {{ coord.lng }}, Role: {{
                coord.role }}
          </li>
        </ul>
      </li>
    </ul>
  </div>
  <ng-template #noMissions>
    <p>Nessuna missione completata.</p>
  </ng-template>
</div>
```

4.3.2 MapComponent

MapComponent è cruciale nel contesto della nostra applicazione in quanto è il componente che si occupa si visualizzare la mappa e monitorare

la posizione del drone in tempo reale durante l'esecuzione di una missione. Di seguito vediamo le parti più importanti e il loro funzionamento.

Il componente gestisce vari aspetti della missione:

- **Stato della Missione**: utilizza variabili booleane come `isPaused`, `shouldStopDrone` e `missioneFinished` per controllare lo stato corrente della missione e poter agire di conseguenza.
- **Variabili di Tracciamento**: `homeBase`, `track` e `dronePath` mantengono rispettivamente la base di partenza, il percorso pianificato e il percorso effettivo seguito dal drone.

Inizializzazione della Mappa e delle Missioni

Nel metodo `ngOnInit()` la mappa viene inizializzata utilizzando il servizio **MapService** e viene fatta la sottoscrizione al servizio **MissionControlService** per ricevere il comando che viene premuto.

Nel metodo `ngAfterViewInit()`, quindi dopo che la pagina è stata renderizzata, il componente si sottoscrive al servizio **MissionService** per ricevere la missione corrente da eseguire.

```
ngOnInit() {
  this.map = this.mapService.getMap();
  this.commandSubscription = this.missionControlService.missionCommand.
      subscribe(command => {
    switch (command) {
      case 'pause':
        this.pauseDrone();
        break;
      case 'stop':
        this.stopDrone();
        break;
      case 'resume':
        this.resumeDrone();
        break;
      default:
        console.log('Unknown command:', command);
      }
    });
}
ngAfterViewInit() {
```

```
  this.missionSubscription = this.missionService.currentMission.subscribe(
      mission => {
    if (mission) {
      this.mission = mission;
      this.startDrone(mission);
    }
  });
}
```

Avvio e Gestione del Drone

Per gestire l'avvio e il movimento del drone per la durata della missione viene utilizzato il metodo `StartDrone(mission: Mission)`. Di seguito spieghiamo le parti principali di questo metodo.

All'inizio del metodo, come vediamo dal codice sotto, prendiamo le coordinate che ci vengono fornite dalla missione che viene passata in input, e attraverso l'uso della libreria `leaflet.motion` viene disegnata la *polyline* che rappresenta il percorso del drone.

```
startDrone(mission: Mission) {
  // path that the drone is supposed to follow
  if (mission.coordinates && mission.coordinates.length) {
    const newCoords = mission.coordinates;
    (L as any).polyline(newCoords, {
      color: '#E49C94'
    }, {
      auto: true
    }).addTo(this.map);
```

Utilizzando il metodo `getDronePath(coordinate)` del **PathService** calcoliamo il percorso effettivo che il drone seguirà. Calcoliamo anche il waitingTime, il tempo di attesa per i punti obiettivo e lo facciamo tenendo conto nel tipo e del numero delle rilevazioni che il drone deve effettuare.

```
      // path that the drone actually follows
    this.dronePath = this.pathService.getDronePath(newCoords);
    const waitingTime = (mission.numberOfReadings * mission.waitingTime) *
        1000;
```

La funzione ricorsiva `nextSegment()` controlla il movimento del drone segmento per segmento. Nel dettaglio questa funzione di occupa di diverse cose:

- **Condizioni di termine**: in quanto funzione ricorsiva occorre controllare quando smettere le chiamate alla funzione, e quindi fermare il drone, e questo avviene attraverso la prima condizione `If`. Questa condizione viene verificata se: la variabile booleana `shouldStopDrone` risulta `true`, oppure si è raggiunta l'ultima coordinata. In tal caso, chiamata `finishMission()` e termina la funzione.

- **Movimento del drone**: questa sezione identifica il `currentPoint` e il `nextPoint` nel percorso del drone, ne calcola la durata utilizzando `getDuration` e crea una polyline animata tra i due punti. Se `nextPoint` è un obiettivo viene settato un timeout in cui la funzione si stoppa per il `waitingTime` calcolato precedentemente, così da permettere al drone di effettuare le rilevazioni. Una volta terminato il timer la funzione riprende con il segmento successivo.

  Se il punto non è un obiettivo passa direttamente al segmento successivo richiamando la funzione `nextSegment`.

```
const nextSegment = () => {
  if (this.shouldStopDrone || currentIndex >= this.dronePath.length - 1)
       {
    this.finishMission();
    return;
  } else {
    const currentPoint = this.dronePath[currentIndex];
    const nextPoint = this.dronePath[currentIndex + 1];
    const duration = this.getDuration();

    this.currentPolyline = this.createPolyline(currentPoint, nextPoint,
        duration, this.map, () => {
      if (nextPoint.role === CoordinateRole.Goal) {
        //show drone marker when the drone is stationary at the goal
        const droneMarker = L.marker([nextPoint.lat, nextPoint.lng], {
          icon: this.droneIcon
        }).addTo(this.map);
        setTimeout(() => {
```

```
            droneMarker.remove(); // remove the stationary drone marker
                after 2 seconds
            currentIndex++;
            nextSegment();
          }, waitingTime);
        } else {
          currentIndex++;
          nextSegment();
        }
      });
      this.currentPolyline.motionStart();
    }
  };
  nextSegment();
 }
}
```

Gestione del Movimento e dell'Animazione

La funzione `createPolyline()` viene chiamata ad ogni ricorsione della funzione `nextSegment()`, quindi per ogni coppia di coordinate viene creata e gestita l'animazione della linea che rappresenta il movimento del drone sulla mappa.

I dati necessari per la creazione del *polyline* vengono passati in input alla funzione, tranne il bearing. Questo dato, che rappresenta l'orientamento del drone, viene calcolato con la funzione `getBearing()`. La funzione prende i due punti del segmento e utilizzando il plugin `Turn.js`, viene calcolato l'orientamento. Dopo aver calcolato il bearing si effettua una correzione del suo valore, nel nostro caso di -48 gradi, per fare in modo che l'icona punti a nord e in questo modo il bearing risulti accurato.

Questa funzione chiama anche il metodo `fillMissionDetails()` del servizio **MissionService** per riempire con i dati della missione, nella sezione dedicata alle informazioni in tempo reale.

Alla fine della creazione del *polyline* viene chiamata la funzione `onComplete()`, che gli viene passata in input. Questa funzione è stata spiegata nella sezione precedente, cioè la sezione che discuteva il controllo effettuato sui nodi per verificarne il ruolo.

```
// create the subsequent polyline and move the drone along the path
createPolyline(start: Coordinate, end: Coordinate, duration: number, map: L.
    Map, onComplete?: () => void) {
  const angle = this.getBearing(start, end);

  // fillMissionDetails fills the mission details for the info box
  this.missionService.fillMissionDetails(this.mission, angle);
  const bearing = angle - 48; // -48 to adjust the icon to point to north

  const polyline = (L as any).motion.polyline([[start.lat, start.lng], [end.
      lat, end.lng]], {
    color: '#0081F3'
  }, {
    auto: true,
    duration: duration,
    easing: (L as any).Motion.Ease.linear
  }, {
    removeOnEnd: true,
    showMarker: true,
    icon: L.divIcon({
      className: 'leaflet-div-icon',
      html: `<i class='fa fa-solid fa-location-arrow fa-2xl' aria-hidden='
          true' style='transform: rotate(${bearing}deg);'></i>`,
      iconSize: L.point(27, 23),
      iconAnchor: L.point(13, 8),
    })
  }).addTo(map);

  if (onComplete) {
    polyline.on((L as any).Motion.Event.Ended, onComplete);
  }
  return polyline;
}
```

Pausa e Ripresa della Missione

I metodi `pauseDrone()` e `resumeDrone()` permettono di mettere in pausa e riprendere l'animazione del drone utilizzando i comandi forniti dal plugin *leaflet.motion*. Questi metodi eseguono un controllo sull'esistenza del `currentPolyline` e che il drone non sia già in pausa, utilizzando la variabile booleana `isPaused`, che viene modificata in base allo stato della missione.

```
pauseDrone() {
  if (this.currentPolyline && !this.isPaused) {
```

```
    this.currentPolyline.motionPause();
    this.missionService.fillMissionDetailsDroneStopped(this.mission);
    this.isPaused = true;
  }
}
resumeDrone() {
  if (this.currentPolyline && this.isPaused) {
    this.currentPolyline.motionResume();
    this.isPaused = false;
  }
}
```

Conclusione della Missione

Il metodo `finishMission()` viene chiamato quando la missione è completata o interrotta, gestendo la pulizia delle risorse e la presentazione dei risultati finali chiamando il metodo che restituisce l'indirizzo per visualizzare il PDF.

```
stopDrone() {
  this.shouldStopDrone = true;
  if (this.currentPolyline) {
    this.currentPolyline.motionStop();
    this.currentPolyline.remove();
    this.currentPolyline = null;
    this.dronePath = [];
    this.coordinatesService.clearCoords();
    this.missionService.fillMissionDetailsDroneStopped(this.mission);
    this.missionService.deleteMission();
  }
  this.isPaused = false;
}

finishMission() {
  this.pdfUrl = this.missionService.completeMission();
  this.missionService.fillMissionDetailsDroneStopped(this.mission);
  console.log('mission finished');
}
```

4.4 Sviluppo dei Servizi Principali

4.4.1 AuthService

`AuthService` è un servizio creato per gestire l'autenticazione degli utenti all'interno dell'applicazione. Utilizza i servizi di `Router` e `CookieService` per gestire la navigazione e memorizzazione delle informazioni di autenticazione. Di seguito verranno spiegati i metodi che compongono questo servizio.

- **login(username, password)**: gestisce il processo di login di un utente, accetta come parametri due stringhe che sono l'username e la password, inserite nel form del Login e, se esistono, salva questi dati nei cookies. Dopo di che naviga verso la Home Page.

```
login(username: string, password: string) {
  if(username && password) {
    this.cookieService.set('username', username);

    this.router.navigate(['']);
  }
}
```

- **logout**: Gestisce il processo di logout dell'utente eliminando il cookie che conteneva l'username dell'utente collegato all'applicazione. Dopo di che naviga verso la Home Page.

```
logout() {
  this.cookieService.delete('username');
  this.router.navigate(['']);
}
```

- **getUser**: Restituisce l'username dell'utente autenticato.

```
getUser() {
  return this.cookieService.get('username');
}
```

- **isAuth**: verifica se l'utente è autenticato, controllando il cookie e restituisce `true` se l'utente è autenticato, altrimenti restituisce `false`

```
isAuth():boolean {
  return this.cookieService.check('username');
}
```

4.4.2 CoordinatesService

Questo servizio è utilizzato per gestire le coordinate dei punti sulla mappa del magazzino di riferimento. Questo servizio mantiene lo stato delle coordinate, aggiunge, aggiorna e rimuove i marker sulla mappa e gestisce le modifiche del ruolo delle coordinate.

Variabili

newCoords è un BehaviorSubject, una specie di Observable, che mantiene un array di coordinate e permette di osservarne i cambiamenti.

markers è una `Map` che associa ad ogni coordinata il suo marker rappresentato da `L.CircleMarker` che è un tipo di marker rotondo fornito da Leaflet.

Funzioni Principali

setHomeBase: utilizza **MapService** per recuperare le coordinate associate alla mappa e dopo imposta la home base del drone come inizio e fine dell'elenco delle coordinate.

```
setHomeBase() {
  const homebaseLatLng = this.mapService.getHomeBase();
  const homeBase = { lat: homebaseLatLng[0], lng: homebaseLatLng[1], role:
      CoordinateRole.Point };
  let currentCoords = this.newCoords.value;
  // Add home base at the beginning and at the end of the array
  if (currentCoords.length === 0 || (currentCoords.length > 0 && (
      currentCoords[0].lat !== homeBase.lat || currentCoords[0].lng !==
      homeBase.lng))) {
    currentCoords = [homeBase, ...currentCoords.filter(c => c.lat !==
        homeBase.lat || c.lng !== homeBase.lng), homeBase];
    this.newCoords.next(currentCoords);
  }}
```

addCoord: aggiunge una nuova coordinata alla sequenza e aggiunge un marker sulla mappa per la nuova coordinata.

```
addCoord(coord: Coordinate, map: L.Map) {
  const currentCoords = this.newCoords.value;
  // Add new coordinate before the last one to mantain the home base at the
      end
  const newCoords = [...currentCoords.slice(0, currentCoords.length - 1),
      coord, currentCoords[currentCoords.length - 1]];
  this.newCoords.next(newCoords);
  this.addMarker(coord, map);
}
```

updateCoord: aggiorna una coordinata specificata dall'indice e aggiorna anche il marker associato a quella coordinata.

```
updateCoord(index: number, updatedCoord: Coordinate, map: L.Map) {
  const currentCoords = this.newCoords.value;
  if (index >= 0 && index < currentCoords.length) {
    // make sure the updated coordinate is formatted correctly
    const formattedUpdatedCoord = {
      ...updatedCoord,
      lat: Number(updatedCoord.lat),
      lng: Number(updatedCoord.lng)
    };
    // update the coordinate in the array
    currentCoords[index] = formattedUpdatedCoord;
    this.newCoords.next(currentCoords);
    // update the marker on the map
    this.updateMarker(currentCoords[index], formattedUpdatedCoord, map);
  }
}
```

getCoords: restituisce un Observable che emette le coordinate aggiornate.

getCoord: Restituisce una specifica coordinata basandosi sull'indice passato in input.

getCoordsSnapshot: Restituisce un'istantanea delle coordinate attuali.

clearCoord: cancella tutte le coordinate.

```
getCoords() {
  return this.newCoords.asObservable();
}
getCoord(index: number) {
  return this.newCoords.value[index];
```

```
}
getCoordsSnapshot(): Coordinate[] {
  return this.newCoords.getValue();
}
clearCoords() {
  this.newCoords.next([]);
}
```

removeCoord: rimuove una coordinata dall'elenco e aggiorna i marker sulla mappa.

```
removeCoord(index: number, map: L.Map) {
  const currentCoords = this.newCoords.value;
  const coord = currentCoords[index];
  // Impedisci la rimozione della home base all'inizio e alla fine
  if (index > 0 && index < this.newCoords.value.length - 1) {
    const updatedCoords = this.newCoords.value.filter((_, i) => i !== index)
        ;
    this.newCoords.next(updatedCoords);
  }
  this.updateAllMarkers(map);
}
```

changeRole: aggiorna il ruolo di una coordinata da *Goal* a *Point* e viceversa e poi aggiorna i marker perchè in base al ruolo hanno un colore diverso.

```
changeRole(index: number, map: L.Map) {
  const currentCoords = this.newCoords.value;
  if (index >= 0 && index < currentCoords.length) {
    const coord = currentCoords[index];
    const newRole = coord.role === CoordinateRole.Point ? CoordinateRole.
        Goal : CoordinateRole.Point;
    const updatedCoord = { ...coord, role: newRole };
    // update array and marker
    currentCoords[index] = updatedCoord;
    this.newCoords.next(currentCoords);
    this.updateMarker(coord, updatedCoord, map);
  }
}
```

addMarker: aggiunge un marker alla mappa per una coordinata e lo memorizza in `markers`.

updateMarker: aggiorna la posizione e il colore di un marker esistente o ne crea uno nuovo se non esiste.

updateAllMarkers: rimuove tutti i marker e li ricrea basandosi sulle coordinate attuali.

removeMarkers: rimuove tutti i marker dalla mappa.

removeMarker: rimuove un marker specifico dalla mappa e da `markers`.

```
addMarker(coord: Coordinate, map: L.Map) {
  const color = coord.role === CoordinateRole.Point ? 'blue' : 'green';
  const marker = L.circleMarker([coord.lat, coord.lng], {
    radius: 5,
    color: color
  }).addTo(map);
  this.markers.set(coord, marker);
}
updateMarker(oldCoord: Coordinate, newCoord: Coordinate, map: L.Map) {
  // find existing marker using the old coordinate
  const marker = this.markers.get(oldCoord);
  if (marker) {
    // update the marker's position
    marker.setLatLng(new L.LatLng(newCoord.lat, newCoord.lng));
    // update the marker's color
    const color = newCoord.role === CoordinateRole.Goal ? 'green' : 'blue';
    marker.setStyle({ color: color });
    // update the marker in the map
    this.markers.delete(oldCoord);
    this.markers.set(newCoord, marker);
  } else {
    // if the marker doesn't exist, add a new one
    this.addMarker(newCoord, map);
  }
}
updateAllMarkers(map: L.Map) {
  this.markers.forEach(marker => marker.remove()); // remove all markers
  this.markers.clear(); // clear the markers array
  this.newCoords.value.forEach(coord => this.addMarker(coord, map)); // add
      all the coordinates to the map
}
removeMarkers() {
  this.markers.forEach(marker => marker.remove());
}
removeMarker(coord: Coordinate, map: L.Map) {
  const marker = this.markers.get(coord);
  if (marker) {
    marker.remove();
    this.markers.delete(coord);
  }
}
```

4.4.3 MapService

Il servizio **Map Service** fornisce funzionalità per inizializzare la mappa, caricare dati relativi alla mappa da un file JSON e gestire eventi click sulla mappa.

Funzioni principali

getMap: carica i dati della mappa con getData(), inizializza la mappa con un sistema di coordinate semplice, in quando essendo indoor non abbiamo latitudine e longitudine. Aggiunge un overlay dell'immagine della mappa e lo adatta ai bound definiti. Aggiunge un marker per identificare la home base del drone e infine restituisce la mappa come oggetto Map di leaflet.

```
getMap() {
  this.getData();
  this.map = L.map('map', {
    crs: L.CRS.Simple
  });
  L.imageOverlay(this.map_image, this.bounds).addTo(this.map);
  this.map.fitBounds(this.bounds);
  const homeBaseLatLng = L.latLng(this.homeBase[0], this.homeBase[1]);
  L.marker(homeBaseLatLng, {
    icon: this.homeBaseIcon
  }).addTo(this.map);
  return this.map;
}
```

getData: carica l'immagine della mappa, le coordinate della home base e i bounds della mappa dal file JSON dedicato al magazzino specifico.

```
getData() {
  const image = coordinates.features[0].properties.image;
  if (typeof image === 'undefined') {
    throw new Error("L'immagine della mappa non e' definita.");
  } else {
    this.map_image = image;
  }
  if (coordinates.features[NUMBER_OF_HOME_BASE].geometry.coordinates.length
      > 0) {
    const coords = coordinates.features[NUMBER_OF_HOME_BASE].geometry.
        coordinates;
    this.homeBase = [(coords as number[])[0], (coords as number[])[1]];
```

```
  }
  if (coordinates.features[0].geometry.coordinates.length > 0) {
    const coords = coordinates.features[0].geometry.coordinates;
    if (Array.isArray(coords[0]) && Array.isArray(coords[1])) {
    this.bounds = [coords[0] as [number, number], coords[1] as [number,
        number]];
    }
  }
}
```

addMapClickListener: aggiunge un listener per l'evento click sulla mappa. Il callback specificato viene chiamato ogni volta che si verifica un click sulla mappa.

removeMapClickListener: rimuove tutti i listener per l'evento click sulla mappa.

```
addMapClickListener(callback: (e: L.LeafletMouseEvent) => void) {
  this.map.on('click', callback);
}
removeMapClickListener() {
  this.map.off('click');
}
```

4.4.4 MissionControlService

Il servizio **MissionControlService** è progettato per gestire i comandi per interagire in tempo reale con il drone che sta eseguendo la missione.

Subject e **BehaviorSubject** da rxjs[9] vengono utilizzati per comunicare lo stato della missione e i comandi tra i componenti dell'applicazione.

```
private missionCommandSource = new Subject<string | null>();
missionCommand = this.missionCommandSource.asObservable();

private isPausedSource = new BehaviorSubject<boolean>(false);
isPaused$ = this.isPausedSource.asObservable();
```

sendCommand emette un nuovo comando tramite `missionCommandSource`. Questa funzione viene chiamata internamente dai metodi `pauseMission`,

[9] **rxjs (Reactive Extension for Javascript**: libreria JavaScript che utilizza gli Observables per lavorare con la programmazione reattiva, la quale gestisce chiamate dati asincrone, callback e programmi basati su eventi

`resumeMissione` e `stopMission` che hanno il compito di aggiornare lo stato di pausa e inviano il loro tipo di comando.

```
sendCommand(command: string) {
  this.missionCommandSource.next(command);
}
pauseMission() {
  this.isPausedSource.next(true);
  this.sendCommand('pause');
}
resumeMission() {
  this.isPausedSource.next(false);
  this.sendCommand('resume');
}
stopMission() {
  this.isPausedSource.next(false);
  this.sendCommand('stop');
}
```

4.4.5 MissionDataService

Il servizio `MissionData Service` gestisce le operazioni CRUD (Create, Read, Update, Delete) per le missioni utilizzando il `localStorage` del browser in modo da memorizzare e recuperare le missioni tra le sessioni dell'applicazione.

saveMission: prende in input come parametro una missione, recupera tutte le missione esistenti utilizzando `getMissions`, aggiunge la nuova missione alla lista, converte la lista in JSON e li salva nel `localStorage`.

getMissions: recupera tutte le missioni e le converte da JSON in un array di missioni.

deleteMission: prende l'id di una missione come parametro in input e rimuove la missione corrispondere dal `localStorage`. Per farlo recupera tutte le missioni, trova l'indice e, se la missione esiste, la rimuove dall'array e aggiorna il `localStorage` con la lista delle missioni aggiornata.

```
saveMission(mission: Mission) {
  const missions = this.getMissions();
  missions.push(mission);
  localStorage.setItem(this.missionsKey, JSON.stringify(missions));
  console.log('Mission saved:', mission);
```

```
}
getMissions() {
  const missionsJSON = localStorage.getItem(this.missionsKey);
  return missionsJSON ? JSON.parse(missionsJSON) : [];
}
deleteMission(missionId: string) {
  const missions = this.getMissions();
  const index = missions.findIndex((mission: Mission) => mission.id ===
      missionId);
  if (index > -1) {
    missions.splice(index, 1);
    localStorage.setItem(this.missionsKey, JSON.stringify(missions));
    console.log('Mission deleted:', missionId);
  }
}
```

4.4.6 MissionService

Il servizio `MissionService` gestisce le missioni in corso e completate, fornendo funzionalità per creare, completare, eliminare e ripetere missioni. Inoltre, aggiorna dinamicamente i dettagli della missione. Utilizza BehaviorSubject per tenere traccia delle missioni e i loro dettagli.

missionSource è un BehaviorSubject che tiene traccia della missione corrente, inizializzato con *null* perchè all'avvio non c'è una missione attiva.

currentMission è un Observable derivato da `missionSource` che permette ai componenti di sottoscriversi così da reagire ai cambiamenti della missione corrente.

missionDetailsSource è un BehaviorSubject che tiene traccia dei dettagli della missione corrente, come velocità, orientamento e altitudine del drone.

missionDetailsSource è Un Observable derivato da `missionDetailsSource` che permette ai componenti di sottoscriversi e reagire ai cambiamenti dei dettagli della missione.

completedMissions è un array che tiene traccia di tutte le missioni completate.

```
private missionSource = new BehaviorSubject<Mission | null>(null);
currentMission = this.missionSource.asObservable();
```

```
private missionDetailsSource = new BehaviorSubject<MissionDetails | null>(
    null);
currentMissionDetails = this.missionDetailsSource.asObservable();

private completedMissions: Mission[] = [];
```

loadCompletedMissions: chiamata all'interno del costruttore, carica le missioni completate dal `localStorage` all'avvio del servizio.

```
loadCompletedMissions() {
  const storedMissions = localStorage.getItem('missions');
  if (storedMissions) {
    this.completedMissions = JSON.parse(storedMissions);
  }
}
```

getMissionById: riceve in input il parametro ID e restituisce una missione completata specifica con l'ID ricevuto.

createMission: prende come parametro in input un oggetto di tipo Missione e crea una nuova missione con un ID univoco creato con il metodo `generateMissionId()`. In fine imposta la nuova missione come missione corrente.

deleteMission: elimina la missione corrente importando `missionSource` a null.

generateMissionId: genera un ID univoco per una missione utilizzando una combinazione di caratteri casuali.

```
getMissionById(missionId: string): Mission | undefined {
  return this.completedMissions.find(mission => mission.id === missionId);
}
createMission(mission: Mission) {
  const newMission = {
    ...mission,
    id: this.generateMissionId(),
    startDate: new Date()
  }
  this.missionSource.next(newMission);
  console.log('Mission created:', newMission);
}
deleteMission() {
  this.missionSource.next(null);
}
```

```
private generateMissionId() {
  return Math.random().toString(36).substring(2, 15) + Math.random().
      toString(36).substring(2, 15);
}
```

completeMission completa la missione corrente importando la data di fine e salvandola nelle missioni completate. Inoltre genera un rapporto in PDF e reimposta la missione corrente a *null*.

```
completeMission() {
  const currentMission = this.missionSource.getValue();
  if (currentMission) {
    currentMission.endDate = new Date(); // Set the end date of the mission
    this.completedMissions.push(currentMission);
    this.missionDataService.saveMission(currentMission);
    const blobUrl = this.pdfService.generateMissionReport(currentMission);
    this.missionSource.next(null); // Reset the current mission
    console.log('Mission completed:', currentMission);
    return blobUrl;
  } else {
    console.log('No mission to complete');
    return null;
  }
}
```

rerunMission: ripete una missione completata creando una nuova missione con un nuovo ID e una nuova data di inizio, quindi naviga alla pagina della nuova missione che gestisce l'inserimento dei dati della missione che si vuole rieseguire nel form della nuova missione.

```
rerunMission(missionId: string) {
  const missionToRerun = this.completedMissions.find(mission => mission.id
      === missionId);
  if (missionToRerun) {
    const newMission = {
      ...missionToRerun,
      id: this.generateMissionId(),
      startDate: new Date(),
      endDate: undefined
    };
    this.createMission(newMission);
  }
  this.router.navigate(['/new-mission', missionId]).catch(err => {
    console.error('Navigation error:', err);
  });
}
```

fillMissionDetails: aggiorna i dettagli della missione corrente, inclusi velocità, orientamento e altitudine del drone.

fillMissionDetailsDroneStopped: imposta i dettagli della missione corrente quando il drone è fermo, azzerando velocità, orientamento e altitudine.

```
fillMissionDetails(mission: Mission, orientation: number) {
  this.missionDetailsSource.next({
    velocity: mission.droneSpeed,
    orientation: Math.floor(orientation),
    altitude: this.getDroneAltitude(),
    start: mission.startDate || new Date(),
    duration: undefined,
    end: mission.endDate
  });
}

fillMissionDetailsDroneStopped(mission: Mission) {
  this.missionDetailsSource.next({
    velocity: 0,
    orientation: 0,
    altitude: 0,
    start: mission.startDate || new Date(),
    duration: undefined,
    end: mission.endDate
  });
}
```

4.4.7 PathService

Il servizio `PathService` è progettato per gestire e calcolare il percorso che un drone seguirà su una mappa. Il servizio crea un percorso basato su una serie di coordinate, calcola la distanza tra i punti, e introduce deviazioni casuali per simulare un movimento più realistico del drone.

`SEGMENT LENGTH` definisce una lunghezza per ciascun segmento intermedio del percorso.

`MAXIMUM DEVIATION` definisce la deviazione massima casuale applicata alle coordinate intermedie per simulare il movimento più realistico.

```
const SEGMENT_LENGTH = 10;
const MAXIMUM_DEVIATION = 0.5;
```

getDronePath: calcola il percorso del drone basato sulle coordinate prese in input, suddivide il percorso in segmenti intermedi di lunghezza `SEGMENT LENGTH`, introduce le deviazioni casuali e infine restituisce un array di coordinate, molto più lungo dell'elenco iniziale, che rappresentano il percorso effettivamente effettuato dal drone.

```
getDronePath(coordinates: Coordinate[]) {
  const dronePath: Coordinate[] = [];
  for (let i = 0; i < coordinates.length - 1; i++) {
    const startCoord = coordinates[i];
    const endCoord = coordinates[i + 1];
    const distance = this.getDistanceInPixels([startCoord, endCoord]);
    const intermediatePoints = Math.floor(Math.round(distance /
        SEGMENT_LENGTH));

    for (let j = 0; j < intermediatePoints; j++) {
      const t = j / intermediatePoints;
      const xIntermediate = Math.round(startCoord.lat + t * (endCoord.lat -
          startCoord.lat) + Math.random() * 2 * MAXIMUM_DEVIATION -
          MAXIMUM_DEVIATION);
      const yIntermediate = Math.round(startCoord.lng + t * (endCoord.lng -
          startCoord.lng) + Math.random() * 2 * MAXIMUM_DEVIATION -
          MAXIMUM_DEVIATION);

      dronePath.push({ lat: xIntermediate, lng: yIntermediate, role:
          CoordinateRole.Point });
    }
    dronePath.push({ lat: endCoord.lat, lng: endCoord.lng, role: endCoord.
        role });
  }
  return dronePath;
}
```

4.4.8 PdfService

Il servizio `PdfService` è responsabile della generazione del PDF del report per una missione specifica. Utilizza jsPDF per creare il documento PDF e include immagini e letture di temperatura basate sui dati della missione. Ecco una spiegazione dettagliata del servizio e dei suoi metodi principali.

generateMissionReport: riceve un oggetto *Mission* in input, genera un nuovo documento PDF e inizia aggiungendo le informazioni generali della missione, come l'ID, il nome e la descrizione, utilizzando `doc.text`. Per tenere traccia della coordinata verticale del documento, per inserire i componenti del documento nel posto giusto, viene utilizzata la variabile `y`.

Dopo le informazioni generali, ogni coordinata viene inserita nel documento e ne viene controllato il ruolo. Se il ruolo è `Goal` allora dopo la coordinata occorre aggiungere anche le rilevazioni fatte, nel caso di fotografie viene usato `addImages()`, per le temperature invece `addTermperatures()`.

Infine restituisce un url del blob del PDF generato.

```
generateMissionReport(mission: Mission) {
  const numberOfReadings = mission.numberOfReadings;
  const doc = new jsPDF();
  const pageHeight = doc.internal.pageSize.height;
  let y = 10; // y-coordinate for the first coordinate

  doc.text('Mission Report', 10, y);
  y += 10;
  doc.text('Mission ID: ' + mission.id, 10, y);
  y += 10;
  doc.text('Mission Name: ' + mission.name, 10, y);
  y += 10;
  doc.text('Mission Description: ' + mission.description, 10, y);
  y += 10;
  doc.text('Mission Start Time: ' + mission.startDate, 10, y);
  y += 10;
  doc.text('Mission End Time: ' + mission.endDate, 10, y);
  y += 10;
  doc.text('Coordinates: ', 10, y);

  mission.coordinates.forEach(coordinate => {
    if (y + 10 > pageHeight) {
      doc.addPage();
      y = 10;
    }
    doc.text('Coordinate: Lat: ' + coordinate.lat + ', Lng: ' + coordinate.
        lng + ', Role: ' + coordinate.role, 10, y);
    y += 10;

    if (coordinate.role === CoordinateRole.Goal) {
      switch (mission.type) {
        case 'camera':
```

```
          y = this.addImages(doc, this.getImages(mission.numberOfReadings),
              y, pageHeight);
          break;
        case 'temperature':
          y = this.addTermperatures(doc, this.getTemperatures(mission.
              numberOfReadings), y, pageHeight);
          break;
        case 'thermal-camera':
          y = this.addImages(doc, this.getThermalImages(mission.
              numberOfReadings), y, pageHeight);
          break;
      }
    }
  });

  const blob = doc.output('blob');
  const blobUrl = URL.createObjectURL(blob);
  console.log('Mission report generated:', blobUrl);
  return blobUrl;
}
```

addImages: aggiunge le immagini al PDF tenendo conto dell'altezza delle immagini e se l'immagine non sta nello spazio rimanente nella pagina allora ne viene generata una nuova. Infine viene restituito y aggiornato alla nuova posizione della pagina.

addTemperatures: allo stesso modo di `addImages()`, aggiunge le temperature rilevate come testo e, se non stanno nello spazio rimanente, viene generata una nuova pagina. Anche in questo caso viene restituito y aggiornato.

```
addImages(doc: jsPDF, images: string[], startY: number, pageHeight: number)
    {
  let y = startY; // inizia dalla posizione Y fornita
  const imageHeight = 90; // altezza dell'immagine

  images.forEach((image, index) => {
    if (y + imageHeight > pageHeight) {
      doc.addPage(); // aggiungi una nuova pagina
      y = 10; // riposiziona l'asse Y all'inizio della nuova pagina
    }
    doc.addImage(image, 'JPEG', 10, y, 100, imageHeight);
    y += 100; // aggiorna la posizione Y per la prossima immagine
  });
  return y; // ritorna la posizione Y aggiornata
```

```
}
addTermperatures(doc: jsPDF, temperatures: number[], startY: number,
    pageHeight: number) {
  let y = startY;
  temperatures.forEach((temperature, index) => {
    if (y + 10 > pageHeight) { // altezza prevista per la temperatura
      doc.addPage();
      y = 10;
    }
    doc.text('Temperature ' + (index + 1) + ': ' + temperature + 'C', 10, y)
        ;
    y += 10; // aggiorna la posizione Y per la prossima temperatura
  });
  return y; // ritorna la posizione Y aggiornata
}
```

Capitolo 5

Interazioni con il Team e Validazione

Per lo sviluppo di ***InDrone*** è stato adottato un metodo di sviluppo *agile*, al fine di garantire un processo di sviluppo costante e sicuro. Questo approccio è stato applicato su tutte le fasi di sviluppo dell'applicazione: dalla definizione dei requisiti, alla pianificazione e definizione delle interfacce, fino alla progettazione e implementazione dell'applicazione.

Il metodo *agile* prevedeva incontri periodici con il team di **Indoor Monitoring** del **TII**. Solitamente ci incontravamo settimanalmente per analizzare i risultati dello sprint precedente, definire gli obiettivi per lo sprint e fissare la data del incontro successivo. In alcune occasioni l'intervallo tra gli incontri si estendeva a due settimane per adattarsi alle esigenze dei membri del team ma anche al tipo di obiettivi fissati, in quanto alcuni obiettivi potevano avere bisogno di più tempo per essere raggiunti.

Questi incontri erano organizzati attraverso Microsoft Teams, dove fissavamo le call di volta in volta.

L'adozione del metodo agile ha permesso di:

- **Rispondere in modo flessibile ai cambiamenti**: Grazie agli sprint brevi e alle revisioni periodiche, è stato possibile adattare rapidamente

il progetto alle nuove esigenze e ai feedback ricevuti, garantendo una continua evoluzione dell'applicazione.

- **Incrementare la collaborazione e la comunicazione**: Gli incontri regolari e la trasparenza del processo hanno facilitato una comunicazione efficace tra i membri del team, assicurando che tutti fossero allineati sugli obiettivi e sulle priorità.

- **Migliorare la qualità del prodotto**: L'approccio iterativo e incrementale ha consentito di testare e migliorare costantemente l'applicazione, identificando e risolvendo tempestivamente eventuali problemi o bug.

- **Ottimizzare la gestione del tempo e delle risorse**: La suddivisione del lavoro in sprint ha permesso di pianificare meglio le attività e di gestire in modo più efficiente le risorse disponibili, garantendo il rispetto delle scadenze e degli obiettivi prefissati.

5.1 Mockup

Dopo aver definito i requisiti per l'applicazione, è iniziato il percorso per identificare l'interfaccia che si adattasse meglio ai requisiti e offrisse un'esperienza d'uso fluida e intuitiva.

Il primo mockup per l'interfaccia dell'applicazione è stato uno disegno fatto a mano che, come possiamo vedere in Figura 5.1, rappresentava la home page dell'applicazione. Questa interfaccia racchiude uno degli scopi principali dell'applicazione, cioè quello di monitorare la missione in corso e interagire con essa.

Il primo disegno rappresentava un buon inizio, ma il consiglio ricevuto è stato quello di utilizzare un software specifico per la progettazione di mockup. Pertanto, per i mockup successivi è stato utilizzato l'editor **Figma**, già menzionato nella sezione 3.4, che ha permesso di definire delle interfacce in

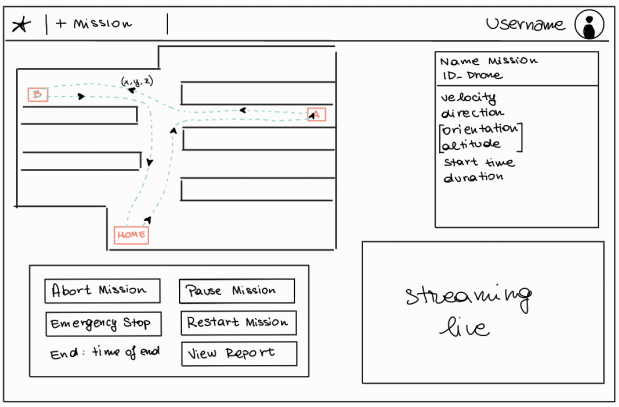


Figura 5.1: Primo sketch per la home page

modo molto più preciso e realistico rispetto a quelli che erano gli obiettivi che si volevano ottenere dall'applicazione.

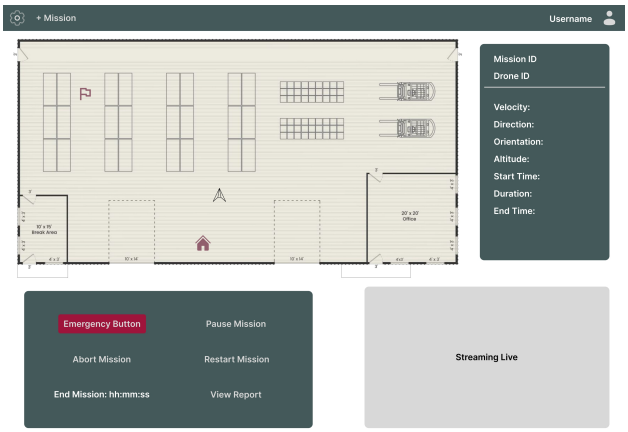


Figura 5.2: Primo mockup con Figma

Il secondo mockup realizzato (Figura 5.2), ha evidenziato alcuni aggiustamenti da apportare:

- La mappa doveva essere più grande, poichè rappresenta il focus dell'applicazione e doveva occupare il maggior spazio possibile.
- Dal punto di vista del design, i pulsanti dovevano essere più semplici e intuitivi, invece che delle scritte sarebbe stato meglio trovare delle icone che rispecchiavano la funzione dei rispettivi pulsanti.

In seguito ai feedback ricevuti, il mockup successivo e definitivo presenta un'interfaccia più pulita ed elegante, ma sempre intuitiva grazie all'utilizzo di icone. La mappa si trova al centro della pagine e più grande, così da essere al centro dell'attenzione dell'utente. (Figura 5.3)

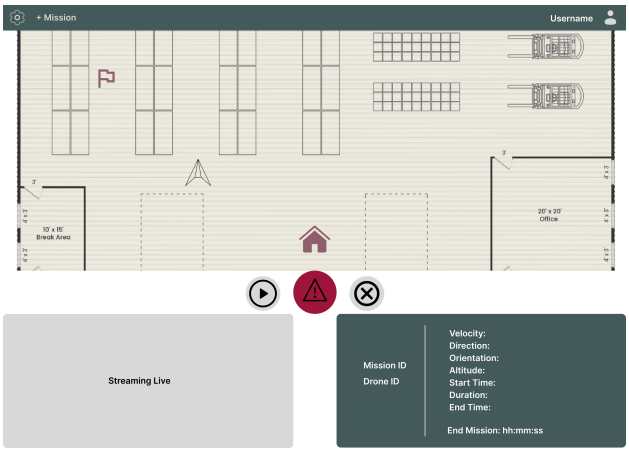


Figura 5.3: Interfaccia definitiva per la home page

Oltre alla realizzazione dell'ultimo mockup della home page, sono stati creati i mockup delle altre interfacce, tra cui quella per la creazione delle missioni, che rappresenta un'altra interfaccia cruciale dell'applicazione. (Figura 5.4)

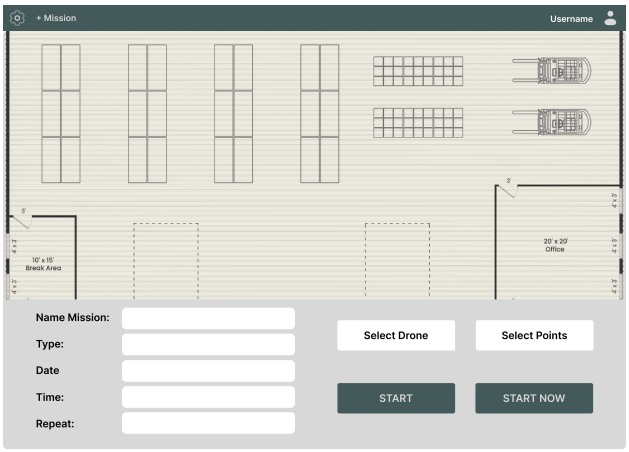


Figura 5.4: Interfaccia definitiva per la pagina per creare una nuova missione

Una volta presentate queste interfacce, sono state approvate, permettendo di proseguire con la fase successiva dello sviluppo.

5.2 Tecnologie

Prima di arrivare al vero e proprio sviluppo dell'applicazione è stata dedicata particolare attenzione alla scelta delle tecnologie da utilizzare per soddisfare al meglio i requisiti funzionali e non funzionali del progetto. Di seguito vengono illustrate le principali tecnologie adottate e le motivazioni delle scelte effettuate.

5.2.1 Angular vs React

Durante la fase iniziale del progetto, è stata valutata la scelta del framework per lo sviluppo dell'interfaccia utente. Il team di **Indoor Monitoring** del **TII** aveva espresso una preferenza per Angular, ma mi era stata lasciata la libertà di utilizzare React, se mi era più familiare o lo avessi ritenuto più adatto alle esigenze del progetto.

Angular e React sono entrambi framework robusti e popolari per lo sviluppo di applicazioni web, ma presentano alcune differenze chiave. Angular, è un framework completo che offre una soluzione out-of-the-box per molte funzionalità comuni, tra cui il data binding, la gestione dello stato, e le direttive. React, è una libreria più leggera e flessibile, che si concentra principalmente sulla creazione di componenti UI, lasciando allo sviluppatore la scelta delle librerie per altre funzionalità.

Dopo una attenta valutazione, abbiamo deciso di utilizzare Angular per i seguenti motivi:

- **Struttura e organizzazione del codice**: Angular incoraggia una struttura del progetto ben definita e modulare, che rende il codice più organizzato e manutenibile.

- **Two-Way Data Binding**: Angular supporta il binding bidirezionale dei dati, semplificando la sincronizzazione tra il modello e la vista. Questo ha migliorato l'efficienza dello sviluppo e ridotto la complessità del codice.

- **Documentazione e Community**: Angular dispone di una documentazione completa e dettagliata, oltre a una vasta comunità di sviluppatori che contribuiscono al suo mantenimento. Questo ha reso più semplice trovare soluzioni a problemi comuni e adottare best practice consolidate.

5.2.2 Leaflet vs MapBox

Inizialmente, avevamo considerato l'utilizzo di **Mapbox** o di **MapLibre**, la sua controparte open source, per la gestione delle mappe all'interno dell'applicazione. Entrambe le soluzioni offrivano caratteristiche avanzate e una buona documentazione, rendendole scelte valide per la visualizzazione di mappe complesse.

Mapbox è noto per la sua potenza e flessibilità, supportando una vasta gamma di funzionalità cartografiche avanzate. **MapLibre**, d'altra parte, essendo una versione open source di Mapbox GL JS, offre molte delle stesse funzionalità senza i costi associati a Mapbox, rendendolo un'opzione interessante dal punto di vista economico.

Tuttavia, dopo una valutazione approfondita, abbiamo optato per l'uso di **Leaflet** per i seguenti motivi:

- **Semplicità e Leggerezza**: Leaflet è una libreria open source leggera e facile da utilizzare. La sua semplicità ha permesso di integrarla rapidamente nel progetto e di personalizzare facilmente le funzionalità necessarie.

- **Flessibilità per mappe indoor**: Creare mappe indoor con Leaflet si è rivelato più semplice rispetto a Mapbox o MapLibre. La struttura

leggera di Leaflet ha facilitato la personalizzazione e l'adattamento delle mappe alle esigenze specifiche del progetto.

- **Personalizzazione**: Leaflet offre ampie possibilità di personalizzazione attraverso plugin e componenti aggiuntivi. Questo ha permesso di adattare le mappe alle specifiche esigenze del progetto senza dover ricorrere a soluzioni proprietarie.

- **Comunità e supporto**: Leaflet ha una comunità attiva e un'ampia documentazione, che rende facile trovare soluzioni a eventuali problemi e integrare nuove funzionalità.

5.3 Sviluppo

Durante l'implementazione dell'applicazione, come detto in precedenza, è stato utilizzato un metodo *agile*, che ci ha permesso di mantenere un flusso di lavoro iterativo e collaborativo. Abbiamo stabilito incontri regolari con il team di **Indoor Monitoring** del **TII**, durante i quali analizzavamo i risultati dello sprint precedente, definivamo gli obiettivi per lo sprint successivo e fissavamo la data per il prossimo incontro. Questo approccio ha garantito una comunicazione costante e una rapida adattabilità ai cambiamenti.

Trello

Per gestire i compiti, le attività e soprattutto la loro importanza, abbiamo utilizzato **Trello** (Figure 5.5), una piattaforma versatile per il project management. Trello ci ha permesso di:

- **Organizzare i compiti**: Abbiamo creato schede per ogni attività, suddividendole in colonne che rappresentavano le diverse fasi del lavoro (ad esempio, "TODO", "DOING"e "DONE").

- **Monitorare il progresso**: Grazie alle etichette e ai commenti, potevamo tenere traccia dello stato di avanzamento di ciascuna attività e discutere eventuali problemi o aggiornamenti.

- **Pianificare gli sprint**: Utilizzando le diverse colonne, per ogni sprint venivano messi i compiti che si pianificavano di completare nella colonna "DOING". Alla fine dello sprint si valutava cosa era stato fatto e cosa invece era rimasto nella colonna ancora da fare e si pianificava lo sprint successivo.

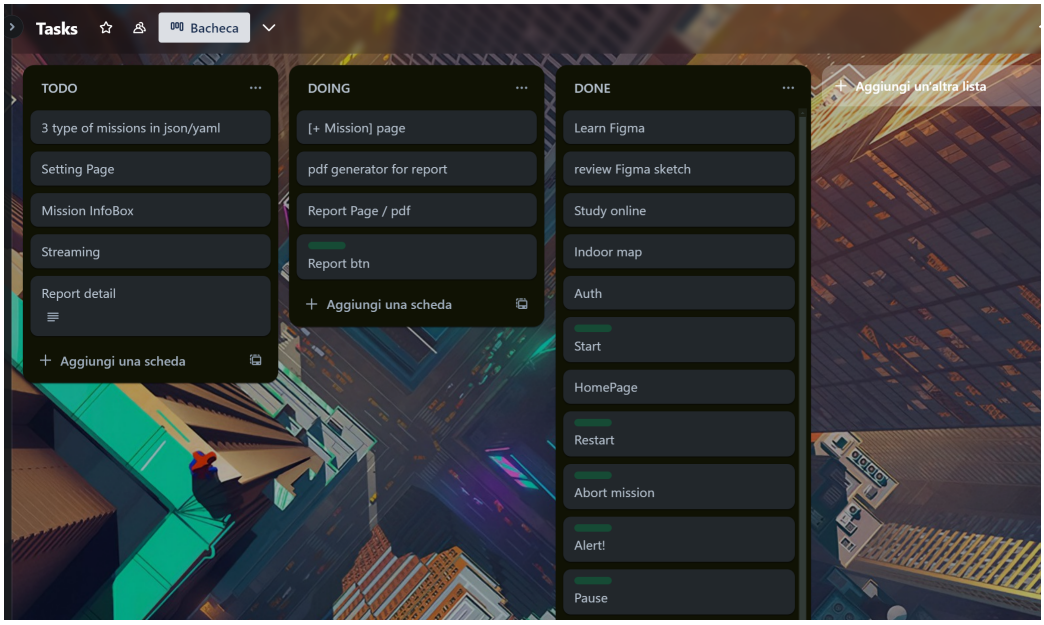


Figura 5.5: Pagina di Trello per la divisione dei Task

Github

Per il controllo delle versioni e la gestione del codice sorgente, abbiamo utilizzato **GitHub**. Questo strumento ci ha offerto numerosi vantaggi:

- **Tracking delle modifiche**: Grazie al sistema di versionamento, potevamo tenere traccia di tutte le modifiche apportate al codice, ripri-

stinare versioni precedenti e documentare le modifiche con messaggi di commit dettagliati.

- **Pull Request**: Le pull request ci hanno permesso di revisionare il codice prima di integrarlo nel branch principale, assicurando una qualità elevata e riducendo il rischio di bug.

- **Issues e Milestones**: Abbiamo utilizzato le issues per segnalare bug, discutere nuove funzionalità e tracciare i progressi. Le milestones ci hanno aiutato a organizzare e pianificare il lavoro per raggiungere gli obiettivi degli sprint.

Grazie all'utilizzo di **Trello** e **GitHub**, abbiamo potuto validare ogni fase dello sviluppo dell'applicazione con il team. Questo ha permesso di correggere tempestivamente eventuali problemi, apportare modifiche necessarie e identificare errori prima del completamento del progetto, anziché alla fine. La validazione continua dei risultati ha garantito che l'applicazione si evolvesse in linea con le aspettative e i requisiti del team, migliorando significativamente l'efficienza e la qualità del processo di sviluppo.

5.4 Validazione della WebApp

Una volta completata ***InDrone***, il **Team di Robotica** ha richiesto un test dell'applicazione, seppur in versione alpha, per valutare l'elaborato. Questo test finale aveva l'obiettivo di verificare se tutti i requisiti definiti all'inizio della nostra collaborazione fossero stati soddisfatti e di individuare eventuali aree di miglioramento.

Questo test è stato effettuato dal team, utilizzando la mappa di un magazzino di proprietà del **TII** e con dati presi da una reale missione effettuata da un drone all'interno dello stesso. Inserendo e adattando questi dati hanno potuto testare l'applicazione per valutarne il funzionamento.

I risultati di questo test finale hanno confermato che tutti i requisiti funzionali dell'applicazione sono stati soddisfatti. L'applicazione ha dimostrato

di operare in modo previsto, offrendo le funzionalità richieste dal **TII** e in specifico dal **Team di Robotica**.

Tuttavia è stato evidenziato che vi è ancora spazio per miglioramenti in diverse aree. Tra i punti negativi emersi vediamo ad esempio che il sistema non è ancora in grado di integrarsi con i veri sistemi utilizzati da TII. Questo rappresenta un ostacolo significativo per l'adozione completa dell'applicazione nel contesto operativo della struttura. Inoltre, la mancanza di un sistema server-side ha comportato alcune difficoltà, limitando le capacità dell'applicazione e riducendone anche la sicurezza (es. login gestito con i cookies), rendendo il sistema molto vulnerabile agli attacchi esterni.

Nonostante ciò, come applicazione web, il progetto ha dimostrato di essere valido e funzionale, rappresentando una solida base per futuri sviluppi e integrazioni.

In conclusione, il test finale ha confermato la validità dell'applicazione sviluppata, pur riconoscendo la necessità di ulteriori miglioramenti e integrazioni per raggiungere una piena operatività all'interno dei sistemi TII.

Conclusioni

La tesi si concentra sull'implementazione e sviluppo di un'applicazione web che permette di creare e monitorare missioni di droni in un spazio indoor. Nella fase iniziale del progetto è stata condotta una ricerca sulla robotica con particolare attenzione ai droni con applicazione indoor. Sono state identificate delle possibili tecnologie che avrebbero potuto essere usate come soluzioni per il nostro obiettivo. Lo studio di queste possibili soluzioni e l'analisi delle loro potenzialità e delle loro limitazioni ci ha permesso di comprendere meglio le esigenze del Team di Robotica di TII e di definire i requisiti dell'applicazione.

Dopo questa prima fase di studio è iniziata la fase di progettazione dell'applicazione. In questa fase sono stati definiti i requisiti funzionali e non funzionali in collaborazione con il Team di Robotica. Una volta definiti i requisiti abbiamo potuto studiare quali potevano essere le interfacce migliori per l'applicazione. Utilizzando il metodo *agile* abbiamo valutato diversi mockup che sono stati iterativamente migliorati e validati nel tempo fino a raggiungere un'interfaccia che fosse intuitiva e che rispondesse alle esigenze del TII e ai requisiti precedentemente definiti.

Anche per lo sviluppo dell'applicazione il metodo *agile* si è rivelato ideale, questo perchè ha garantito un processo di sviluppo iterativo e collaborativo. Abbiamo stabilito incontri regolari con il Team di Robotica durante i quali analizzavamo i risultati ottenuti fino a quel punto e definivamo gli obiettivi per l'incontro successivo. Questo approccio ha garantito aggiornamenti costanti e adattabilità ai cambiamenti.

Infine il progetto è stato approvato e validato dal TII grazie ai test effettuati con la demo. I risultati ottenuti hanno dimostrato che l'applicazione sviluppata soddisfa le funzionalità richieste dal TII e in specifico dal Team di Robotica.

Tuttavia, sono emerse alcune limitazioni durante il progetto. La principale è la mancanza di un'integrazione server-side, che ha ridotto le capacità dell'applicazione e la sua sicurezza. Inoltre, il sistema non è ancora in grado di integrarsi con i veri sistemi utilizzati dal TII, limitando la sua applicabilità nel contesto operativo.

Per migliorare l'applicazione, possibili implementazioni future potrebbero concentrarsi sull'implementazione di un backend dedicato per aumentare la sicurezza e le capacità di integrazione. Inoltre una funzionalità che inizialmente avrebbe dovuto far parte del progetto, ma che non è stata implementata per motivi di tempo e complessità, è lo streaming video in tempo reale dalle telecamere dei droni. L'integrazione dello streaming video insieme alle altre tecnologie di monitoraggio avrebbe migliorato la qualità del monitoraggio delle missioni. Essendo un applicazione web, al giorno d'oggi sarebbe utile considerare lo sviluppo dell'applicazione utilizzando un framework come Electron, progettato per lo sviluppo di applicazioni desktop multipiattaforma.

In conclusione, questo progetto ha dimostrato la fattibilità e l'efficacia di una soluzione web-based per il monitoraggio delle missioni indoor. Nonostante le limitazioni attuali, l'applicazione rappresenta una base solida per futuri sviluppi e integrazioni. Il lavoro svolto ha contribuito significativamente al campo delle tecnologie di monitoraggio e gestione delle missioni, e offre numerose opportunità per miglioramenti e ampliamenti futuri.

Bibliografia

[1] Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, and Francesco Lamonaca. Sensors and measurements for unmanned systems: An overview. *Sensors*, 21(4):1518, 2021.

[2] Rick D Davenport. Robotics. *Smart Technology for Aging, Disability, and Independence: The State of the Science*, pages 67–109, 2005.

[3] Tamás Haidegger, Péter Galambos, and Imre J. Rudas. Robotics 4.0 – are we there yet? In *2019 IEEE 23rd International Conference on Intelligent Engineering Systems (INES)*, pages 000117–000124, 2019.

[4] Pierluigi Sandonnini. Robot: cosa sono e quali sono le previsioni future. `https://www.ai4business.it/robotica/robot-cosa-sono-e-quali-sono-le-previsioni-future/`, 2022.

[5] Angular Team. Angular - architecture overview, 2024.

[6] Angular Team. Introduction to services and dependency injection, 2024.

[7] Francesco La Trofa. Cos'è la robotica, come funziona e quali sono gli esempi di applicazione. 2022.